Volume 09 Issue 02 Published, May 19, 2005 ISSN 1535-864X DOI: 10.1535/itj.0902

Intel® Technology Journal

Compute-Intensive, Highly Parallel Applications and Uses

Smarter computer technologies can rapidly and transparently analyze large complex datasets and explore different outcomes, and ultimately help people do what they want to do more easily. RMS (recognition, mining and synthesis) is the term used by Intel to describe this class of emerging applications. This issue of Intel Technology Journal (Volume 9, Issue 2) discusses these new capabilities or "uses," which are driving a whole new set of computer and architecture requirements.

Inside you'll find the following articles:

More information, including current and past issues of Intel Technology Journal, can be found at:
http://developer.intel.com/technology/itj/index.htm

Volume 09 Issue 02 Published, May 19, 2005 ISSN 1535-864X DOI: 10.1535/itj.0902

Intel® Technology Journal

Compute-Intensive, Highly Parallel Applications and Uses

Articles

THIS PAGE INTENTIONALLY LEFT BLANK

Preface

Compute-Intensive, Highly Parallel Applications and Uses
by Lin Chao
Publisher, *Intel Technology Journal*

"If a man can write a better book, preach a better sermon, or make a better mouse trap than his neighbor, though he build his house in the woods, the world will make a beaten path to his door." The sentence, usually shortened to emphasize the better mouse trap, is by Ralph Waldo Emerson (1803-1882), an American poet and philosopher. What Emerson said so long ago applies even today as inventive people work to build a better computer mouse such as wireless or optical.

In the Research and Development Labs at Intel, we are applying Emerson's philosophy to building smarter computers which are more natural and easier to use—a lot less strict and a lot more adaptive to humans. We're building smarter computer technologies that can rapidly and transparently analyze large complex datasets and explore different outcomes, and ultimately help people do what they want to do more easily. These new capabilities or "uses" are driving a whole new set of computer and architecture requirements.

The seven papers in this issue of Intel Technology Journal (Volume 9, Issue 2) focus on Compute-Intensive, Highly Parallel Applications and Uses. They review the exploratory work into complex and large "workloads" that can ultimately run efficiently on future computers. Generally, these are characterized as compute-intensive, highly parallel workloads requiring new levels of intelligence, performance, and sophistication in both hardware and software that does not exist today. And we look at how the performance scalability and uses on parallel architectures of such applications can help to best architect the next generation of computers.

The first paper is on ray tracing, a technique used in photo-realistic imagery such as in the creation of computer games and special digital effects in movies. Ray tracing can be an important workload to establish requirements for new architecture that will one day run efficiently on mainstream computers.

The second and third papers are on computer vision. The vast accumulation of digital data requires new classes of applications. We are investigating computing platforms that can deliver enough performance for these future workloads to enable their use in mass-market applications. Computer Vision (CV) is one such workload. In the second paper we introduce and characterize some of the most common CV algorithms and applications. We chose a complete video surveillance application as a representative case study for a complex CV workload. The third paper looks at Intel's Open Source Computer Vision Library and describes using OpenCV for "learning-based vision," where objects such as faces, or patterns such as roads, are learned and recognized.

The fourth and fifth papers look at data mining, or the ability to extract knowledge, acquire models, and draw meaningful conclusions from a dataset. The fourth paper examines data mining applied to bioinformatics. Bioinformatics is the recording, annotation, storage, analysis, and search/retrieval of gene sequences, protein sequences, and structural information. In this paper, we report on the

performance scalability analysis of six bioinformatics applications on a 16-way Intel® Xeon™ multiprocessor system. The fifth paper looks at large-scale data-mining problems based on tree-based models. Tree-based models, in the context of large-scale data-mining problems, provide many challenges for a computing platform. The balance between complexity and accuracy is studied for different parameter sets and its performance impact is discussed.

The sixth paper looks at optimization algorithms using the Interior Point Method (IPM). IPM has become a dominant choice for solving large optimization problems for many scientific, engineering, and commercial applications. In this paper we describe a parallel IPM for solving optimization problems.

The seventh paper examines future IT enterprise platform requirements based on usages and deployment models. We present the needs of various vertical industries (e.g., retail, manufacturing, financial) and discuss the business usage and the technology deployment trends across these industries. We describe how the emerging models are different in their characteristics from those prevalent today, and, using several real-world examples, explain the platform implications.

These papers look at new, intelligent, large, and sophisticated workloads that can analyze large complex datasets and explore different outcomes, and ultimately help people do what they want to do more easily on computers. We also look at vertical industries and how their needs will steer platform definitions. These capabilities or "uses" are driving a whole new set of computer and architecture requirements. Let's experiment together on future usage models impacting future computer platforms to build tomorrow's smarter computers.

Foreword

Recognition, Mining and Synthesis

By:
Bob Liang
Director, Application Research Lab, Corporate Technology Group

Pradeep Dubey
Senior Principal Engineer and Manager, Innovative Platform Architecture,
Corporate Technology Group

Intel's RMS (recognition, mining and synthesis) taxonomy[1] offers a way to describe a class of emerging applications. This issue of Intel Technology Journal (Vol 9, Issue 2) discusses a small subset of RMS applications to help the reader understand the nature of such applications. In turn, the reader will understand the high-level platform requirements for these workloads and the implications for processor platforms of tomorrow.[2] The technology underlying these applications is likely to have broad applicability to a wide range of emerging applications with mass appeal in various market segments including digital enterprise, digital home, and digital health.

The wave of digitization is all around us. While none of us has a crystal ball to predict the future "*killer app"* (any new application with universal appeal), it is our belief that the next round of applications will be about solving the *data explosion* problem for end-users, a problem of growing concern for both enterprise and home users. Digital content continues to grow by leaps and bounds in various forms, including unstructured text on the web; digital images from consumer cameras to high-definition medical images; streams of network access logs or e-Commerce transactions; and digital video data from consumer cameras and surveillance cameras. Add to this massive virtual reality datasets and complex models capable of interactive and real-time rendering, and approaching photo-realism and real-world animation.

Recognition is a type of machine learning which enables computers to model objects or events of interest to the user or application. Given such a model, the computer must be able to search or *mine* instances of the model in complex, often massive, static or streaming datasets. *Synthesis* is discovering "what if" cases of a model. If an instance of the model doesn't exist, a computer should be able to create it in a virtual world.

Beyond its use as a taxonomy, RMS offers an integrated view of underlying technologies. Traditionally we have treated "R," "M," and "S" components as independent application classes. For example, graphics (a form of synthesis application), computer vision, and data mining are traditionally considered independent, stand-alone applications. However, an integration of these component technologies, if achieved real-time in an *iRMS* (interactive RMS) loop, may lead to exciting new usages. For example, consider a *virtual dressing room* which lets you use an archive of apparel and images, and create various synthetic combinations of these, or a further extension to

richer forms of real-time reality augmentation. Processor platforms of today still have a long way to go before the compute power reaches the required level for these applications, which in many cases go well beyond teraflops. However, it is our belief that this dawn of tera-era[3] has an unprecedented value proposition to the end user in terms of significantly increased visual realism, and productivity in the face of the digital data explosion.

[1] Dubey, Pradeep. A Platform 2015 Model: Recognition, Mining and Synthesis Moves Computers to the Era of Tera. Feb. 2005.

[2] Borkar, S.; Dubey, P.; Kahn, K.; Kuck, D.; Mulder, H.; Pawlowski, S.; Rattner, J. Platform 2015: Intel Processor and Platform Evolution for the Next Decade. 2005.

[3] Gelsinger, Pat. Architecting the Era of Tera. IDF R&D Keynote Address, Feb. 2004.

Technical Reviewers

Mark Chang, Sales and Marketing Group
Gideon Gerzon, Corporate Technology Group
Radek Grzeszczuk, Corporate Technology Group
Jackson He, Digital Enterprise Group
Igor V. Kozintsev, Corporate Technology Group
Valery Kuriakin, Technology and Manufacturing Group
Chu-cheow Lim, Software and Solutions Group
Joel Munter, Software and Solutions Group
Ara Nefian, Corporate Technology Group
Oscar Nestares, Corporate Technology Group
Dmitry Ragozin, Corporate Technology Group
Adam Seeger, Corporate Technology Group
Gordon Stoll, Corporate Technology Group
Rahul Sukthankar, Corporate Technology Group

THIS PAGE INTENTIONALLY LEFT BLANK

Ray Tracing Goes Mainstream

Jim Hurley, Corporate Technology Group, Intel Corporation

Index words: ray tracing, global illumination, ambient occlusion, immediate mode API, retained mode API, photo-realistic rendering, physically correct photo realistic rendering, occlusion culling

ABSTRACT

We present an introduction to the rendering technique known as "ray tracing." We propose that its performance has reached the stage where it is feasible that it will take over from raster graphics in the near future for interactive gaming and other application domains. We investigate various aspects of ray tracing and compare and contrast them with the raster equivalent. Finally, we analyze ray tracing's platform requirements and scalability potential.

INTRODUCTION

Ray tracing is the act of tracing the trajectory of a ray from one point to another to determine if anything is hit and the distance to the nearest hit point. Although for our purposes ray tracing can be thought of as a "workload," in the larger graphics world, ray tracing is considered to be a tool. Rendering systems use a variety of such tools to achieve their goals. In almost all graphics workloads, the rendering portion consumes >90% of the available resources. Nowadays, more and more of the techniques used in photo-realistic imagery are based on ray tracing.

HOW RAY TRACING IS USED

The following is a partial list of how ray tracing is used:

- Visibility testing is used to determine if there is an unobstructed path from A to B. "Eye rays" are shot from a camera to determine what can be seen. This is known as inverse/reverse ray tracing.
- Illumination testing is used to determine if there is an unobstructed path from A to a light source. This enables us to determine very precise and accurate shadows and illumination.
- Perfect reflection and refraction: subsequent ray trajectory based on the properties of a material struck by a visibility ray. (Very few real materials exhibit such perfect properties.)
- Diffuse or anisotropic reflection and refraction gives a more realistic determination of the consequences of a ray intersecting with a realistic material. Typically, a "shader" is invoked whenever a ray strikes a surface. This shader then determines some distribution and "weights" of subsequent reflected and refracted rays in various directions.
- Light transport determines how light flows from the various light sources in an environment to one or more "cameras." Recall that that we mentioned that "eye rays" are shot from the camera into the scene to determine what is visible. These "forward" rays travel in the opposite direction. Some light will flow in such a way that it does not impact the image seen by the camera (i.e., may or may not bounce off various surfaces in the scene, but nothing that the camera sees is directly or indirectly effected); other light may directly influence the image seen by the camera, and some other light may indirectly influence the image seen by the camera.
- Ambient occlusion is a "trick" used extensively by Pixar and other movie-production studios. An assumption is made that the lighting environment consists of a horizon-to-horizon hemisphere of uniform illumination intensity. Whenever an "eye ray" intersects a surface, some number of visibility rays are shot over a hemisphere sample space. The rays are shot to determine how "exposed" the intersection point is to the "sky." The farther more of the rays travel before hitting anything, the more exposed the intersection point. The "weight" of these feeler rays is used to determine an ambient intensity for the intersection point. Ambient occlusion produces a pleasing look to the image because of its soft intensity transitions. Note that no consideration of material properties, or actual light placement etc. is taken into account. Usually an ambient occlusion map is produced as a result of this pass, and this map is considered the base layer to which other lighting layers are added; sometimes, this pass alone is performed.

Figure 1: Image demonstrating Ambient Occlusion. Copyright © 2003 Pixar [1]

HOW RAY TRACING IS USED WITH OTHER TOOLS

Photo-realistic rendering, the ultimate goal of all graphics, is achieved by using a variety of tools, several of which use ray tracing directly or indirectly. Ray tracing can easily be used to determine the direct illumination of a given scene. Indirect illumination is more difficult, as not only light paths are important, material properties play a significant role also. However, even in indirect illumination, ray tracing techniques can be employed. Some techniques effectively allocate a budget of rays dedicated to forward tracing from light sources; the remainder of the rays are used in the usual fashion of inverse tracing from the camera. Forward ray tracing is used to trace the path of light as it emanates from various light sources, strikes various surfaces in the scene, and reflects, refracts, etc. from surface to surface. As the rays land on various surfaces, material shaders are invoked that determine how the light energy is absorbed, reflected, refracted, scattered, etc. from the surface, and at each such spot the color at that point is stored in a cache. Subsequently, during the inverse tracing phase, the cached photons are effectively treated as a larger collection of light sources. Ray tracing itself does not solve the problem of creating *photo-realistic* images; however, it is an important tool that is used extensively in conjunction with a wide variety of other tools.

True photo-realistic rendering requires solving a "global illumination" problem, namely that everything in a scene affects everything else in the scene (to some degree), and that indirect illumination is vitally important, even though it might only have a subtle effect on the final image. It is this subtle effect that makes the difference between a false-looking image and one that looks "real." (The goal is to achieve the effect called "suspension of disbelief"; in other words, the imagery created can be intended to be a cartoon or "live action").

Raster Graphics and Ray Tracing

Ray tracing and "raster graphics" can be used to attempt to solve the above-mentioned global illumination problem. In fact, ray-tracing techniques can achieve the exact same results as raster-based techniques (including all the approximations and tricks that raster solutions typically require); however, it does not work the other way around. Both of these approaches have their advantages and disadvantages, and both work in very different ways with very different system implications which we summarize here.

Raster Graphics

The primary differentiating factor between raster- and ray-tracing approaches is that a ray tracing approach enables one to solve a global problem, while a raster-based approach seeks to achieve similar results by solving a local problem. Raster graphics attempts to render an image efficiently by making certain convenient assumptions. In particular, it treats triangles as if each triangle is entirely independent of every other triangle. Raster graphics hardware is capable of achieving extremely high throughput. However, the triangles are not independent, and in fact, this presumption places a lot of restrictions on the rendering system. Modern raster graphics APIs work in "immediate mode" where there is an expectation that the raster engine renders each triangle or command upon receipt. There is a concept of the current state and the current triangle. This is the Graphics Processing Unit's (GPU) view of the entire world; it has no idea if or what comes next. Modern raster systems are extremely efficient, and some of the above-mentioned limitations can be worked around some of the time: for example, by rendering in multiple passes, or by employing a variety of approximations, tricks, etc. to leverage the tremendous performance of these devices. However, these approximations and multi-pass approaches impose limitations that Independent Software Vendors (ISVs) either have to live with or learn to avoid.

Raster and ray-traced systems have different cost functions and scaling characteristics, listed below. In general, due to the way that raster systems work, they process every pixel of every submitted triangle to determine the final image that needs to be displayed. Raster graphics performance scales strongly with the number of triangles and pixels that have to be processed for a given image, so cost scales roughly linearly with viewport size and overall scene complexity, as follows:

- If a scene requires 100M triangles to be submitted for rastering, and each has an average of 10 pixels, the

raster system has to process 1B pixels. That is if it only takes one rendering pass.

- ISVs very carefully manage the complexity of their content, and various occlusion-culling techniques are used to minimize the quantity of submitted triangles.
- Because geometry must be grossly simplified to avoid the above performance problems, various techniques are used to create the illusion of more detail than what actually exists.
- Texture mapping is used to simulate extra detail. In fact, multiple layers of textures are often used.
- As mentioned earlier, part of the cost function is related to the number of pixels that have to be rendered. Often, multiple textures get applied per pixel, and each layer of texture can require many samples from the texture buffers; consequently, the bandwidth requirements of raster-based solutions can be astronomical.

Raster graphics assumes that the triangles in a scene are independent of each other. This allows hardware to process each triangle independently and even to process multiple triangles and pixels simultaneously. But, in fact, the triangles are not independent: triangles can cast shadows on other objects or other triangles in the same object, but also triangles can be translucent, and they can reflect and refract light, and so on. Raster systems can get around some of these limitations using a variety of tricks. The tricks, such as those listed below, usually work under certain circumstances, and those situations where they fail must be avoided in order to preserve "suspension of disbelief."

- Complex lighting effects, caused by objects reflecting and refracting light, can be simulated by running a real offline global illumination solution and extracting the results and storing them in maps. This can lead to plausible images being generated. However, these light maps are captured at a point in time, and with a particular arrangement of all the lights and objects in a scene, that they do not accommodate dynamic lighting situations.
- The Z buffer is used to perform a binary test to decide if a point on the screen covered by a new triangle is closer to the camera than the same point on the screen covered by a previous triangle. However, this makes the assumption that all triangles are opaque. Where translucency is involved, the translucent objects have to be separated from the rest of the objects, and all the objects need to be rendered in a particular order to avoid artifacts.
- To compensate for lower triangle counts, various bump maps and normal maps are used to create the illusion of increased complexity. Texture maps were originally intended to represent the micro-detailed texture of a surface; however, raster solutions often use texture maps to represent macro-level features. Lower precision models cause lots of problems: silhouette edges are blocky, and it is very difficult for artists to get the look and feel of what they are striving for. When the models are viewed from shallow angles, it becomes apparent that the surface details are not really there; the ISV has to work hard to understand all the limitations and to avoid those situations where these techniques break down.
- Shadows are another issue. Firstly, shadows are critical; images without shadows appear to have objects floating in space. Shadows are important visual clues that help associate an object with the surface that it is on or above, etc. Raster solutions don't really handle shadows because of the independent triangles presumption; instead raster techniques emulate shadows. There are a variety of ways of doing this: some require rendering the scene many times from the point of view of each light, and some involve determining the silhouette edges of an object from a particular light's point of view and casting rays through these silhouette edges to form so-called shadow volumes, etc. Each of these techniques works after a fashion, but all have various artifacts and restrictions. In fact, in one raster-based game we investigated recently, we found that there were five different shadow algorithms in use, and the ISV had to pick which one to use, on an object-by-object basis.

RAY TRACING

Ray tracing, and by extension, global illumination does not suffer from these limitations just outlined, but it has its own set of problems that we discuss here. Firstly, there are fundamental differences between raster and ray-tracing approaches. Because ray tracing takes a global approach, the natural interface to it is different than that used with raster graphics. Recall that raster graphics use an "immediate mode" interface, and is only aware of a current state and a current primitive at any point in time. Ray tracing, on the other hand, needs a "retained mode" interface where random access to the whole scene is required, and when a visible triangle is determined, a specific shader needs to be invoked on demand.

Ray tracing relies on a so-called "acceleration structure"; this is organized as a spatial partitioning or indexing structure. Any given object or scene is decomposed into regions of empty space and finer and finer partitioning of filled space. This structure allows us to efficiently determine the path a ray would take through space, and to test it only against those triangles that would be in the vicinity of the ray's trajectory. Otherwise, we would need to test every ray against every triangle to test for any

possible intersection. If the scene is composed of rigid bodies, or articulated rigid bodies, the acceleration structure can be constructed in advance and loaded into the application along with the scene's geometry. If the scene is dynamic, then the acceleration structure may have to be built on the fly, per frame, which can be very expensive. We have found that kd-trees make the best acceleration structures. If one needs to build an acceleration structure, with a very large model, the algorithm (greatly simplified) goes something like this:

- Read in ***all*** the vertices, think about them for a while, then propose a (single) split based on some cost function. This effectively gives you two sub-trees.
- Repeat the previous step for each and every sub-tree you create until certain termination criteria are met.
- You will notice that, at the higher parts of the tree, traditional caches won't help much. However, once a certain threshold is reached, where the entire sub-tree fits, constructing the lower parts of the tree gets easier and easier.
- Conversely, you can see that if you are given the higher parts of the tree, building the rest of the tree is relatively inexpensive. Unless the object literally comes apart at the seams, the top-most parts of the tree rarely ever change.
- Often, in a gaming scenario, for example, other parts of the application require some sort of spatial partitioning structure (such as the occlusion-culling engine, the physics engine, the collision detection engine, etc.). If all of these used the same structure that the ray-tracing engine requires, even if at lower precision, this would greatly facilitate those cases where the tree needs to be built on the fly.

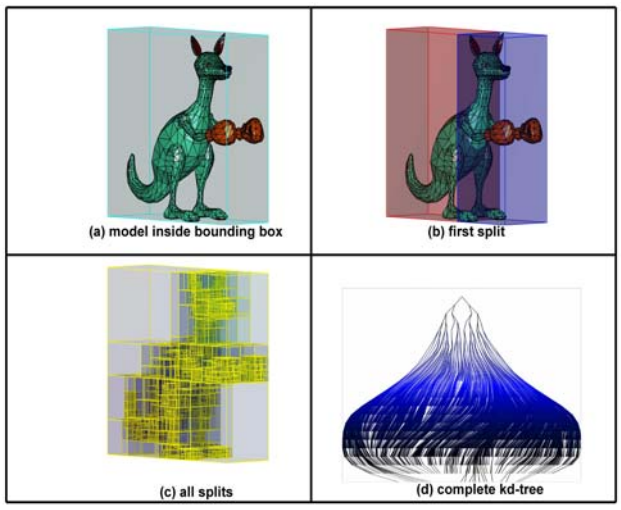


Figure 2: An example of a model, and how it is decomposed into an acceleration structure

Building these trees, and building them well is a huge topic all by itself. We have implemented algorithms that result in incredibly good trees, but it would take a separate white paper to describe the techniques used in detail. Also, separate research has begun in areas related to lazily building such trees (only build what's needed, when it's needed) and building them to a sufficient (i.e., minimal) level of detail. Finally, the very best acceleration structures are built using a cost function that trades off a given platform's computation and memory system's characteristics: these structures need to be built on the platform that they will get used on.

The next issue to tackle is the platform cost of tracing a ray. This requires traversing the acceleration structure in a serialized sequentially dependent fashion until the ray finds a leaf node. Every time we traverse the structure it will be for a different ray, but if you shoot rays in a spatially coherent fashion, most likely the rays will take the same path through the structure. The (simplified) traversal algorithm is as follows:

- Test the ray against the split plane defined for the current volume of space.
  - Perform a simple test (a few simple ops and a compare).
- Determine if the ray goes cleanly to one side or the other of the split plane, or passes through it.
  - Perform a data dependent unpredictable branch.
  - Go to the "left," "right" or both sub-node(s).
- Move onto the next node(s) and repeat.

Once a leaf node is encountered, we need to perform a computationally intensive ray-triangle intersection test for every triangle in the node. Even then, the ray might miss all the triangles there (hence it is best that leaf nodes hug the boundary of an object as tightly as possible). We have optimized these algorithms extensively, and we have optimized the acceleration structures to minimize the number of traversal steps to a leaf node, greedily accounting for as much empty space as possible, and we have minimized the number of triangles in each leaf node, for **any** given platform. Remember, it is cheaper to test a ray against an empty space than to test it against a bunch of triangles in a cell and find out that the ray misses them all. We have figured out how to shoot arbitrary groups of rays as a beam, performing most of the traversal using the beam instead of all the rays in the beam. We use vector approaches where they make sense: testing the four planes that represent the limits of a beam against each split plane, testing the triangles in a leaf node against each ray that gets that far, etc.

The result of using beams is that a lot of the time, the beam finds a very deep "entry point" in the kd-tree for all

the rays in the beam. Often, this entry point is at the leaf node itself, meaning that the bulk of the work is now computational. Also, as we report later, we find that we get really excellent cache hit rates, which dramatically lowers the external bandwidth cost of using ray tracing (assuming traditional CPU-style cache hierarchies).

However, the really exciting news is how the ray tracing workload scales. It is strongly effected by the number of rays shot in a scene and weakly effected by the complexity of the scene, which is different than raster graphics. Recall its performance scales with the number of triangles and pixels rendered, which is a function of the scene complexity and the overall viewport size. In contrast, ray tracing scales linearly with the number of rays shot and only logarithmically with the complexity of the scene. For a fixed resolution image, the cost of raster graphics doubles (roughly) as the complexity of the scene doubles; for ray tracing, you would have to increase the ***viewed*** scene complexity by 10x to double the cost. We have found that even with today's hardware (HW) raster accelerators, a single CPU running software (SW), ray tracing will catch up with the HW raster engine around the 1M triangles per scene mark, and will always outperform it above that. We have also discovered that the performance of ray tracing scales linearly with the number of CPUs. Another observation is that the performance of ray tracing is not so much overall scene complexity dependent, but dependent upon the ***visible*** complexity of the scene. Imagine a world where there are millions of triangles, but only 50K are visible at a time (such as a building). Given the above mentioned beam concept, the ray-tracing algorithm will quickly zero in on just those parts of the overall structure where the visible triangles are, effectively, shrinking the tree to just that part. Therefore, ray tracing performance scales with the observed complexity, rather than the overall complexity. Effectively, the acceleration structure behaves like an infinite level of detail occlusion-culling mechanism.

ISV Implications

ISVs can build complex models, without having to fake details, or have harsh polyganized outlines, silhouettes, etc. Moreover, they do not have to compromise by trading off model and scene complexity against overall performance. And finally, turning on more and more features just results in more rays:

- Visibility is determined by shooting (1 - n) "eye" rays per viewport pixel (depending on how much anti-aliasing is needed).
- Infinitely precise shadows cost 1 ray per eye ray/triangle hit point, per light.
- Exact reflections, refractions are 1 ray each (times the number of subsequent bounces that are permitted).
- For translucency, keep refracting rays through translucent surfaces until an opaque one is hit, or some saturation point is reached.
- For anisotropic reflections/refractions. a budget of subsequent sample rays per surface struck must be allocated, which can be bound by the limits of new rays per bounce, and the number of bounces, etc.
- For global illumination, use more rays: forward rays and regular rays. In essence, everything boils down to rays.

All of the above work exactly as expected: there are no corner cases where they don't. The cost of a scene can be calculated very accurately and parameters can be tweaked to hit frame rates. The platform implications boil down to how coherent those rays are: eye rays can be engineered to be highly coherent as can shadow rays. This is less so for reflected and refracted rays. Not only can an ISV budget exactly what he or she can (or cannot) afford given a target platform and other application parameters such as model complexity, etc., but an ISV can safely use these features individually or in combination:

- For one thing, shadows are no longer an issue: they are on all the time for everything. There is no need for tricks, or hacks. Different algorithms don't need to be chosen selectively for individual objects, and there is no need for multi-pass approaches. Moreover, those shadows are exact, perfect, and always right, and everything shadows everything else: a character's nose casts a shadow on its face, and dimples in the nose have shadows inside them. If the ISV wants a shadow to fall a particular way, he or she can engineer that by selecting a mask to indicate which lights to seek shadows from.
- There are no multiple passes of rendering from virtual cameras behind reflective surfaces, and no need to texture the result into the rendering pass. No environmental maps are needed that only comprehend infinitely distant scenery. And, multiple reflections work–naturally.
- Translucency is no longer a problem. There is no need for sorting of geometry or for careful ordering of what gets rendered first. It all just works. Even if the desire is to have everything be translucent and there are sufficient rays available, this can also be done.
- All of the above-mentioned techniques work together in one pass without the need for complex sorting of what needs to happen first, or for what limitations exist.

Everything else that gets used in raster graphics, such as multiple texture mapping for simulation of fine detail surface texture, still works, and all the sampling and

filtering that goes with that. All the hacks and tricks used to simulate complex lighting like pre-rendered global illumination solutions etc. can still be used. In fact, you can create exactly the same results as a raster platform using ray tracing instead. However, unless *all* the hacks and tricks that simulate complexity, shadows, and complex lighting are used, then fine details will go missing: triangles that appear to have surface detail and shadows, when rastered, will actually appear flat and featureless and have weird colors that don't appear to belong there when ray traced, if all the aforementioned tricks are not used.

RAY TRACING PERFORMANCE

The basic core of a ray-tracing engine is the act of shooting a ray, and that, in turn, depends heavily on three key algorithms: acceleration structure traversal, ray triangle intersection tests, and an arbitrarily complex "shader," invoked if there is a ray-triangle hit. We determined that we needed to establish a "benchmark" by which we could gauge performance: the number of ray-segments per unit time. A ray segment is one leg of the journey of a ray. Each "bounce," if you will, also each ray shot at a light to determine if the current spot is in shadow, is also a ray segment. We determined that overall performance could be characterized in terms of some aggregate number of ray segments per second; there were three main dimensions:

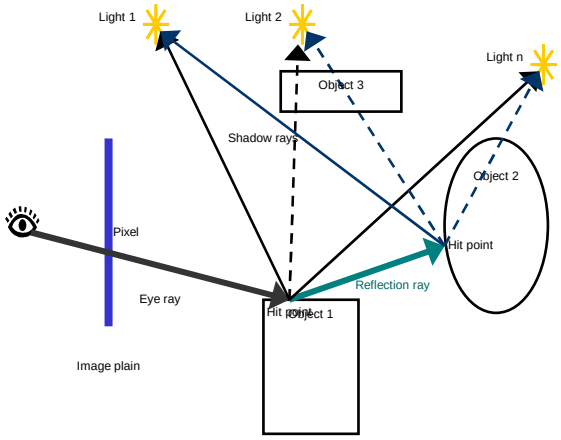


Figure 3: Diagram illustrating how a single pixel's rays decompose into eight ray segments

1. The number of rays shot per pixel–a quality metric. More raysegs per pixel account for more and more lights, bounces, sampling, etc.
2. The number of pixels per frame (grows in discrete steps, 640x740 to 1024x768 …etc.).
3. The number of frames per second (30fps for movies, 75fps for games).

We determined that 450M raysegs/S was the threshold where real-time ray tracing becomes interesting. We assumed a frame rate of 30fps, an image size of 1M pixels, and 15 raysegs per pixel. This could quickly escalate (linearly) if 75fps, 3M pixel displays, and higher quality is taken into account. We measured the computation and raw and external bandwidth required for a variety of scenes (recall that performance is viewport and scene complexity dependant) and found that the per ray segment cost was 1500-3000 FLOPS and 600-1400 raw bytes, with cache hit ratios of 300-1200:1. We were able to achieve up to 100M raysegs/S for simple models and large viewports on desktop machines (a 3.2 GHz Pentium® 4 processor), and measured linear performance scaling up to 128 CPUs in cluster configurations.

Scalability of the Ray Tracing Algorithm

Ray tracing scales strongly with the number of actual rays that need to be traced and weakly with the overall complexity of the scene being rendered. The number of rays that need to be traced depends on the following conditions:

- The raw dimensions of the viewport (i.e., the number of pixels). Over sampling or anti-aliasing would increase the number of rays per pixel shot into the scene.
- The frame rate (24fps for movies, 72-75 Hz for game content).
- The number of active lights. If a ray hits an object, rays are shot from the hit point to all of the active lights to determine if there is an unobstructed path.
- The number of bounces allowed per ray. Some engines place a cap on the number of subsequent bounces caused by reflection or refraction, as each such bounce contributes less and less to the value returned to the eye.
- The sophistication of the shaders invoked when a ray hits a triangle. Most surfaces are not perfect reflectors or refractors; instead a shader invoked at an initial hit point may choose to shoot many secondary rays in a non-uniform distribution, thereby integrating the resulting returned values. Similarly, the first set of secondary rays can spawn its own second set of secondary rays, etc.
- Whether or not global illumination techniques are employed. Some techniques take a budget of rays and shoot them from the lights into the scene letting them bounce around, thereby caching the light intensities of the various hit points. A subsequent traditional ray tracing pass might then consider these cached hit points as additional light sources, for example.

Ray tracing scales weakly with the complexity of the scene because of the acceleration structure. Due to the structure's hierarchical nature, ray tracing reduces the cost of finding a ray triangle intersection to O Log N. As the size of the model increases linearly, the overall size of the acceleration structure can grow linearly also, but the cost of finding an intersection doesn't.

Scalability studies for raster graphics and ray tracing graphics have been performed. In general, the cost of raster graphics processing is linear with the number of pixels to be processed. The cost of ray tracing scales linearly with the number of rays shot, so roughly one can claim that ray tracing performance scales linearly with viewport size.

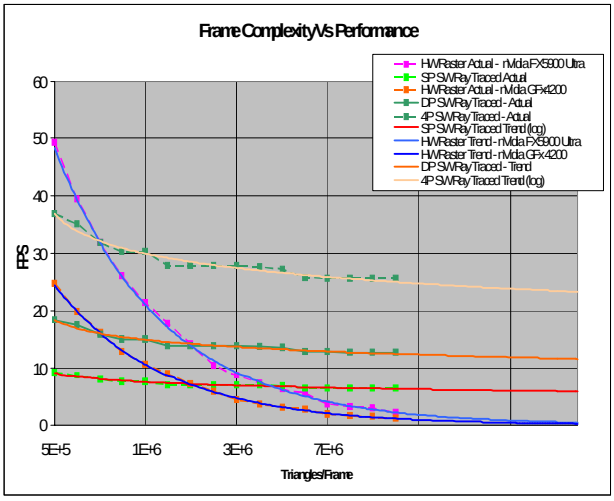


Figure 4: Frame complexity vs. performance of SW ray tracing vs. HW raster

If the viewport size remains fixed, then the ray-tracing performance scales logarithmically with the complexity of the scene. This means that if you compare a HW raster engine and a SW ray-tracing engine using the same input for both engines, although the HW will initially beat the SW, the SW will eventually catch up with the HW. In fact we measured this and found that the intersection point is in the vicinity of the 1M triangle range, i.e., when the scene complexity exceeds 1M triangles, a SW ray-tracing solution will always outperform a HW raster solution.

Figure 4 is a log/linear chart. We created many versions of the same model at various resolutions from 10K to 10M triangles, and we fed them into an nVidia GeForce FX 4200 (lower blue curve) and an nVidia GeForce FX 5900 Ultra (upper blue curve). A SW ray tracer running on the same 3.2 GHz Pentium 4 processor (with a 512 KB L2 cache) was used to drive the HW cards (performance above the 10M triangle mark is extrapolated from curves fitted to the measured data). The curved blue lines show the performance of the HW cards: performance declines at 1/X. The (almost) straight yellow/orange lines represent SW ray-tracing engine performance on a single processor and 2- and 4-way Symmetrical Multi-Processor (SMP) systems. Performance declines at 1/Log(X).

Scene and Visibility Complexity Implications

Figure 5 shows the three scenes used in this analysis of the ray-tracing algorithm. The scenes are carefully chosen to span a wide range of visual complexity. One can see that the level of detail as well as the quality of the images are at least as good as those produced by the high-end raster graphics engine today.

The first scene is a typical bar illuminated with one light. The model consists of about 250K triangles with about 300K vertices. The size of the acceleration structure that represents the spatial distribution of detail is 18 MB, and the scene data itself is 29 MB. The model has 54 different textures, and the texture maps to a total of 30 MB in size. Given the camera placement in this scene, the majority of the room's details are visible. There are no reflections or refractions in this scene. The second scene is a top view of a VW Beetle illuminated with three lights. The model consists of 1.87M triangles with about 4M vertices. The size of the acceleration structure is 127 MB, and the scene data itself is 224 MB. The image uses 20 texture maps totaling 11 MB in size. This particular view of the car was chosen because most of the geometric detail is in the interior of the car. Reflections are enabled with a maximum of two reflections per ray. The third scene is of the inside of a room in UC Berkeley's Soda Hall. The model consists of 2.2M triangles with about 1.6M vertices.

The size of the acceleration structure is 148 MB; the scene data is 226 MB. The model has 20 texture maps totaling 17 MB in size. This model was chosen because despite the fact that the whole model has 2.2M triangles, only a small percentage is visible at any time. If the camera is outside, we can see only the shell of the building; if it is inside the building we can only see the details of that particular area. In this particular room there are three lights. The full model has 1300, but the rest are disabled. Reflections and refractions are disabled.

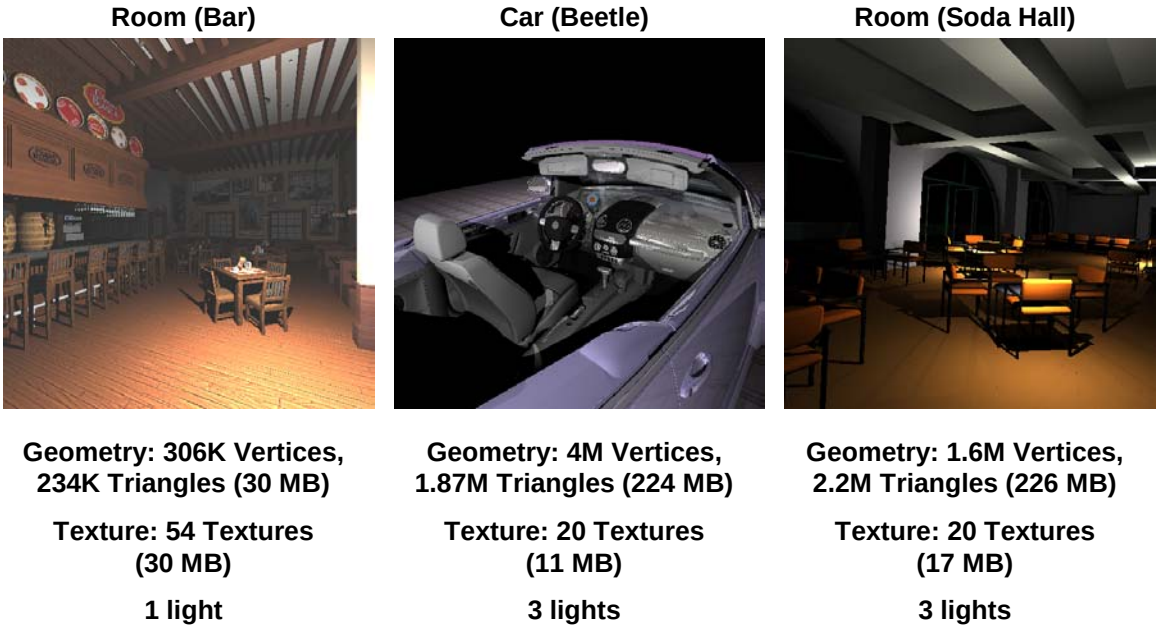


Figure 5: Scenes with different visual complexities used in this ray-tracing analysis

Table 1 shows the raw FLOPs and bandwidth required per ray segment across the various models all rendered at the same 1024x1024 resolution. As you can see, there is some correlation between overall model complexity and required performance as we go from the bar scene (250K Triangles) to the beetle scene (~2M Triangles). Although complexity increases 10x, the computational and bandwidth costs only increase ~2x.

Table 1: Computation and memory requirements

Measured Data	Room (Bar)	Car (Beetle)	Building (Soda Hall)
Flop / RaySeg	1518	2954	1488
Byte / RaySeg	586	1382	793
Byte / Flop	0.386 / 2.59:1	0.467 / 2.14:1	0.533 / 1.87:1

Table 2: Bandwidth at each level of memory hierarchy

Measured Data	Room (Bar)	Car (Beetle)	Building
Core to L1 BW	1,300.0 GB/s	920.0 GB/s	1,260.0 GB/s
L1 to L2 BW	66.5 GB/s	57.6 GB/s	49.5 GB/s
External BW	6.1 GB/s	12.7 GB/s	1.1 GB/s
Raw / Ext BW	216:1	72:1	1141:1

However, as we go from the Beetle to the soda hall, we note that even though both models are about equally complex, because only a fraction of the scene is visible in any frame, the cost is dramatically lower for the soda hall. Table 2 shows the raw and cache filtered bandwidths required for the same scenes assuming each is rendered at 30fps. Here we see that the cache hit rate seems to correlate with the observed model complexity, so the soda hall scene shows the best performance thanks to the beam effect zeroing in on the portion of the acceleration structure that is effectively used by the ray tracing algorithm.

Performance Scalability Studies

Performance data was collected on 2-, 4- and 8-way SMP machines and for large cluster configurations with up to 128 nodes. As you can see from Figure 7, performance scales linearly with the number of processors for the SMP systems, and from Figure 8 you can see a similar story for clusters of systems with up to 128 nodes. We also measured performance with and without hyper-threading on the SMP systems. From Figure 6, you can see that we get a >25% overall performance improvement across the board when hyper threading is turned on.

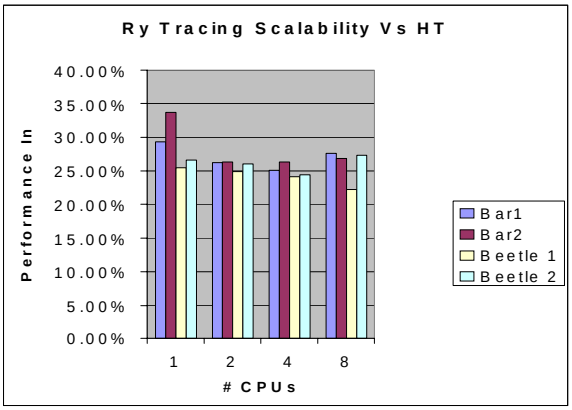


Figure 6: Hyper-threading effect on ray-tracing performance for 1-, 2-, 4- and 8-CPU systems

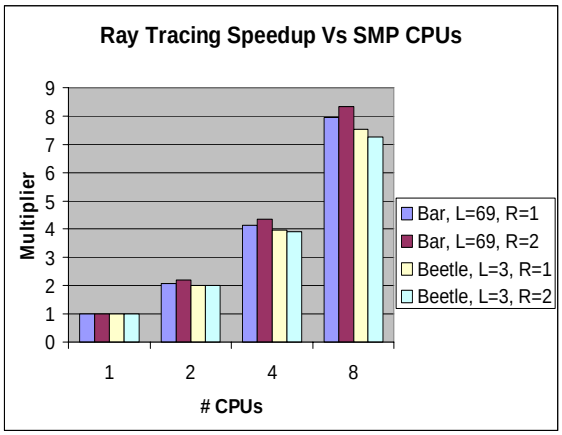


Figure 7: Scaling of ray-tracing performance for 2-, 4- and 8-way SMP machines

(L = Lights, R = Reflections)

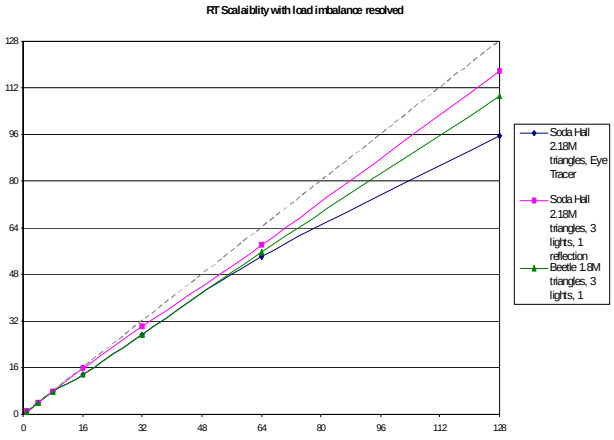


Figure 8: Scaling of ray-tracing performance for cluster systems with 1-128 CPUs

CONCLUSION

Ray tracing has long been considered too expensive for mainstream rendering purposes. Movie production studios have only recently begun the transition to using it; however, the true cost of ray tracing has been very poorly understood until recently. It is now poised to replace raster graphics for mainstream rendering purposes. Its behavior is very well suited to CPU processors, and scales well with hyper threading and multi-processor configurations. The traditional cache hierarchy associated CPUs is very effective at managing the external memory bandwidth requirements. For ISVs, a transition to ray tracing is a huge step forward freeing them from all the limitations imposed on them by today's raster-based approaches. Ray tracing is one tool that can enable ISVs to aspire to achieving high fidelity photo (or cartoon) realistic imagery.

ACKNOWLEDGMENTS

Various people contributed to the materials that went into this report including Gordon Stoll, Alex Reshetov, Victor Lee, and Alexei Soupikov. If I left anyone out, please accept my apologies in advance.

REFERENCE

[1] P.H. Christensen, D.M. Laur, J. Fong, W.L. Wooten, D. Batali, "Ray Differentials and Multiresolution Geometry Caching for Distribution Ray Tracing in Complex Scenes," Eurographics 2003 conference, Computer Graphics Forum, Volume 22, Issue 3 (September 2003).

AUTHOR'S BIOGRAPHY

Jim Hurley is a principal engineer in the Application Research Lab of the Corporate Technology Labs at Intel. He has been working in the area of computer graphics for over 20 years. His e-mail is jim.hurley at intel.com.

THIS PAGE INTENTIONALLY LEFT BLANK

Computer Vision Workload Analysis: Case Study of Video Surveillance Systems

Trista P. Chen, Corporate Technology Group, Intel Corporation
Horst Haussecker, Corporate Technology Group, Intel Corporation
Alexander Bovyrin, Corporate Technology Group, Intel Corporation
Roman Belenov, Corporate Technology Group, Intel Corporation
Konstantin Rodyushkin, Corporate Technology Group, Intel Corporation
Alexander Kuranov, Corporate Technology Group, Intel Corporation
Victor Eruhimov, Corporate Technology Group, Intel Corporation

Index words: computer vision, workload analysis, RMS, video surveillance, foreground detection, estimation-maximization, Gaussian mixture, particle filter, condensation filter, Markov chain Monte Carlo, eigen analysis, singular value decomposition, optical flow, motion estimation

ABSTRACT

The vast accumulation of digital data requires new classes of applications that impact a computer user's life. We are investigating computing platforms that can deliver enough performance for these future workloads to enable their use in mass-market applications. Recognition, Mining, and Synthesis (RMS) are three key classes of workloads that distill enormous amounts of data. Among these is Computer Vision (CV), an important workload that will greatly benefit from future architecture and algorithm innovations.

We illustrate these innovations by introducing and characterizing some of the most common CV algorithms and applications. We focus on (1) algorithms for Gaussian mixture models, (2) particle filtering (condensation filtering), and (3) optical flow/motion estimation, which are key ingredients of many modern CV algorithms. We also discuss computer vision applications, such as video surveillance, autonomous (intelligent) vehicles and driver assistance systems, entertainment and augmented reality, and smart health care.

We chose a complete video surveillance application as a representative case study for a complex CV workload. Video surveillance is one of the most resource-demanding CV applications that has wide-spread application. We analyze an entire pipeline of a video surveillance system to obtain computation and bandwidth characteristics.

Our characterization of individual CV algorithms as well as complete CV systems can be used to guide algorithm researchers to develop new algorithms that run faster on existing and future computing platforms. Furthermore, we hope that it will raise the awareness of the application developers to optimize their programs. It will also provide input data for architects to develop future computing platforms that run these workloads more efficiently.

INTRODUCTION

The amount of data in the world is doubling every three years. This includes data found on the Web, in our personal albums, and digital music collections, etc. However, these data are usually not organized efficiently and not used to their full extent. For example, we might spend hours opening images we have on our hard-drives (assuming they are still stored in the hard-drive without losses) to look for one image of ourselves with a particular person. Looking forward, three fundamental processing capabilities: Recognition, Mining, and Synthesis (RMS), will be the key to future data processing, and Computer Vision (CV) is one of its main workloads.

This paper is organized as follows. In "Trends in Computer Vision Workloads," we discuss the key algorithms and applications of CV workloads, as well as current trends. In "Introduction to the Video Surveillance System," we study in more detail one of the most representative CV workloads: video surveillance. We first

describe the complete pipeline of a video surveillance system and its individual components, and then we present results from workload analysis of the video surveillance system. We conclude with a summary of our findings and future research opportunities.

TRENDS IN COMPUTER VISION WORKLOADS

Key Algorithms of Computer Vision Workloads

Computer Vision (CV) algorithms can be categorized as low-level image-processing techniques and high-level analysis techniques. Image-processing techniques include filtering, and feature extraction, which have been extensively studied and are thus not elaborated on in this paper. High-level analysis usually involves probability-based approaches. We discuss the Gaussian mixture model, and particle filters, since these are used extensively in tracking and motion analysis.

We believe the trend in CV algorithms is moving towards higher level analyses (such as semantics), which make extensive use of probability-based techniques. Probability-based algorithms are our main focus. We start with discussing probability distribution models. We then discuss multi-model data fusion from color images, the integration of color video and range data, and the integration of face images, fingerprint images, and iris images.

Gaussian Mixture Model

In order to track an object in the scene, knowledge about what the object looks like is needed. Such knowledge is described by the statistical distribution of the region of interest (which could be the foreground object or the background). The Gaussian mixture model is used widely to describe the region of interest [1]-[6], and has been incorporated into a variety of algorithms for tracking and recognition in CV.

A Gaussian mixture model with m components is described as a sum over Gaussian probability distributions:

$$p(x|Obj) = \sum_{m=1}^{M} \pi_m N(x, \mu_m, \sigma_m^2 I)$$

where u_m is the mean of component m, σ_m is the variance of component m, and π_m is the weight of component m. By applying the Estimation-Maximization (EM) algorithm, the Gaussian mixture model parameters can be trained.

Particle Filter/Condensation Filter

The particle filtering algorithm is a sequential Monte Carlo method. The algorithm is powerful in approximating non-Gaussian probability distributions, and it has a wide range of applications including object tracking in CV [7][8][9]. Particle filtering is based on sequential importance sampling and Bayesian theory. It models the data distribution by random sample measures composed of particles, that are samples from the space of the unknowns, and their associated weights.

There are two main steps in the algorithm: selection/updating and mutation/prediction. The first step selects the particles for reproduction. The particles representing the most likely parameter candidates are the ones most likely to be selected. During this step, heavy particles generate new ones, while light particles are eliminated. The second step allows each particle to evolve according to a given transition probability kernel. The pictorial description of the algorithm is shown in Figure 1.

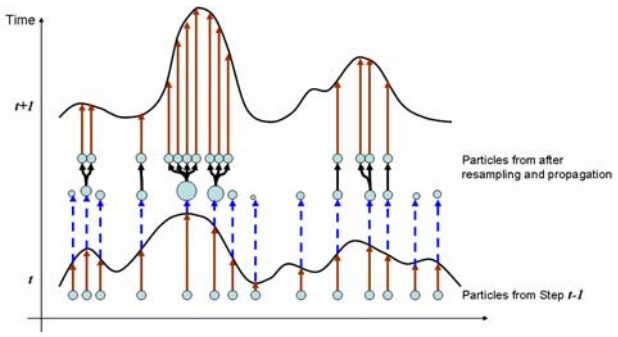


Figure 1: A pictorial description of particle filtering

Similarly, in using random samples to estimate distribution, Markov Chain Monte Carlo (MCMC) is also often used. References can be found in [10][11].

Optical Flow/Motion Estimation

Optical flow estimation is a technique used to compute the apparent image motion field of the scene. Besides tracking objects, the motion field is useful for extracting objects from video sequences. When there is discontinuity in the motion field, it often means there are different depths of pixels and the depth discontinuities may correspond to the outlines of different objects. An introduction to optical flow can be found in [12][13], and its application to road detection can be found in [14].

Key Computer Vision Applications and Opportunities

Digital video recording devices are now ubiquitous in our daily lives. They are mounted indoors in offices, hospitals, and outdoors in parking lots and intersections. Some vehicles even have cameras recording passengers and the surroundings of the car. Massive amounts of video data are collected by these "digital eyes." CV technologies can add intelligence to these digital eyes, adding "brains" to these imaging devices.

With both digital eyes and brains, CV can be a very useful tool: it can be used for video surveillance, entertainment/augmented reality applications, autonomous vehicles and driver assistance systems, robotics, and smart health care. With the rapid growth in CV, we believe more innovation in this field is inevitable. We now discuss these CV applications.

Video Surveillance/Security

Video surveillance addresses real-time observation of humans or vehicles in some environment (indoor, outdoor, or aerial), leading to a description about the activities of the objects with the environment or among the objects. It is used mostly for security monitoring, as well as traffic flow measuring, accident detection on highways, and routine maintenance in nuclear facilities, etc. Interested readers can refer to [17][18].

From a workload analysis perspective, video surveillance is one of the most interesting CV applications. The reason is twofold. First, a complete video surveillance system consists of foreground segmentation, object detection, object tracking, human or object analysis, and activity analysis. It touches many core topics of CV. By understanding and analyzing a complete video surveillance system, we can obtain insights on general CV workloads. Second, video surveillance is currently gaining increasing importance for security applications worldwide. "Traditional" video surveillance systems have been used pervasively in airports, banks, parking lots, military sites, etc. However, to get any useful information from these systems, humans either have to watch a massive amount of video data in real-time with full attention to detect any anomalies, or the video data can only be used as evidence after an abnormal event has occurred, due to the lack of real-time automatic tracking and analysis. Automatic video surveillance, as opposed to traditional video surveillance, adopts CV algorithms such as those mentioned in the last section to alleviate the load on humans and to enable preventative acts when an anomaly is detected. We will use the term video surveillance, instead of automatic video surveillance, hereafter. A comprehensive introduction to video surveillance systems can be found in the next section.

Autonomous Vehicles and Driver Assistance Systems

Driver safety is a very important issue in our lives. CV can serve as a third eye for a driver to enhance the safety of vehicles and of their occupants. Example uses of CV for intelligent vehicles include (1) parking assistance; (2) landmark detection to assist the car in following the road; (3) traffic sign detection and recognition for route planning and alerting the driver; (4) obstacle detection, especially detecting the presence of pedestrians in a driver's blind spot; (5) driver condition monitor for intelligent airbag deployment and to monitor a driver's distraction level [19][20][21].

Figure 2 (from [22]) shows a road detection example. The detection result can be used for the later obstacle detection in knowing the 3D location of the obstacle. Figure 3 from [23] shows an example of how CV can help in locating pedestrians/obstacles. Pedestrian/obstacle detection and tracking is harder than conventional object detection and tracking. Both the camera in the car and the objects are moving. The background changes constantly. Figure 4 (from [24]) shows different poses of a driver for safe airbag deployment as shown in Figure 5 (from [24] as well).

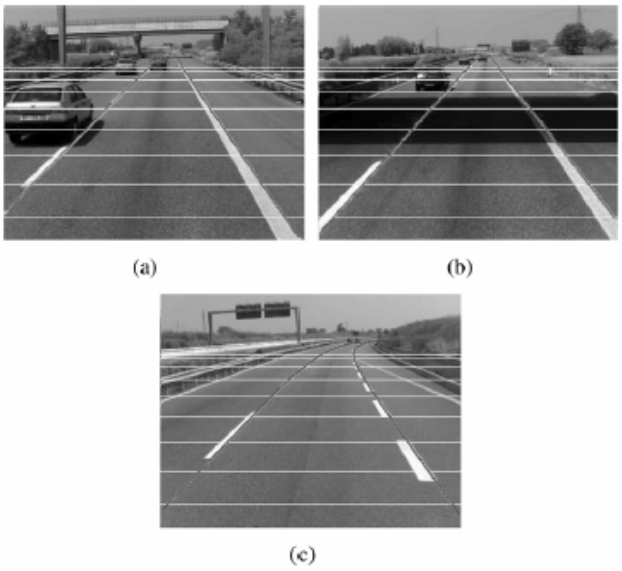


Figure 2: Road-detection result for highways and the 3-D projected model with different conditions. © 2005 IEEE, courtesy of [22]

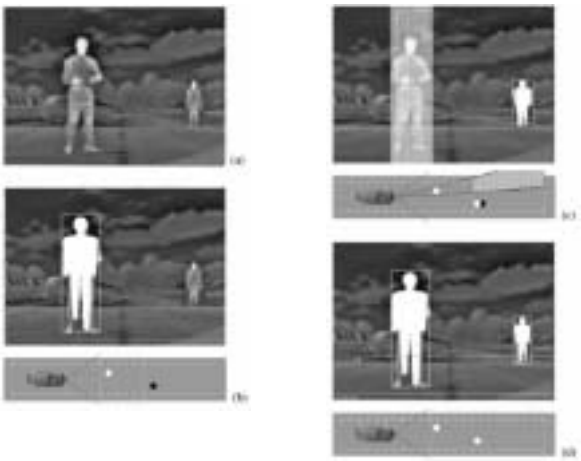


Figure 3: Obstacle detection example: fusion of the results from two resolutions. (a) Input image, (b) results of low-resolution processing, (c) results of original resolution processing, and (d) final fused results. © 2005 IEEE, courtesy of [23]

Figure 4: Example images of occupant script poses. From top left: sitting normally, leaning halfway, leaning completely forward, leaning back, leaning right, leaning left, moving hands about cabin, opening glove box, hands on face, stretching, adjusting radio, hat in lap, putting on hat, moving while wearing hat, removing hat, feet on dashboard. © 2005 IEEE, courtesy of [24]

Figure 5: Occupant position and posture-based safe airbag deployment. © 2005 IEEE, courtesy of [24]

Entertainment/Augmented Reality

Another interesting application of CV techniques is augmented reality, which combines real video content, a real object extracted from the video, and rendered graphical models. That is, three-dimensional virtual objects are embedded in the real video scene. Augmented reality applications can be seen in entertainment scenarios such as games and movies. This technique is useful for 3D manipulation and maintenance tasks, and it is helpful during surgical procedures as clinical data can be overlaid on real video content.

CV techniques are needed to extract the 3D information from the environment, so that the virtual objects can be placed in the proper locations (an example is shown in Figure 6 from [25]). Articulated body tracking can be used to animate the character in the gaming environment (an example is shown in Figure 7 from [26]).

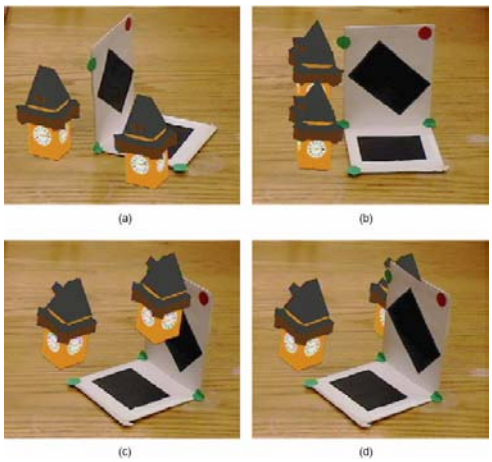


Figure 6: Visible-surface rendering of texture-mapped affine virtual objects. Affine basis points were defined by the centers of the four green dots. The virtual towers were defined with respect to those points: (a) initial augmented view; (b) augmented view after a clockwise rotation of the object containing the affine basis points; (c) hidden-surface elimination occurs only between virtual objects; correct occlusion resolution between physical and virtual objects requires information about the geometric relations between them; (d) real-time visible surface rendering with occlusion resolution between virtual and real objects. © 2005 IEEE, courtesy of [25]

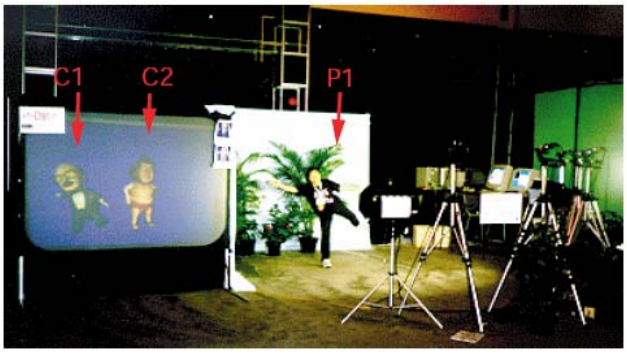


Figure 7: Virtual metamorphosis. The motion of the person (P1) in the dancing area controlled the movement of the tuxedo-wearing cartoon character (C1). A person in another dancing area controlled the movement of the sumo wrestler cartoon (C2). © 2005 IEEE, courtesy of [26]

Smart Health Care

CV can be used in health care for the elderly and the disabled. Human body tracking and activity analysis can help detect anomalies, such as a person falling. Human subjects can also use their hand or body gestures to control the home environment if hand or body tracking is in place [27]. Similar ideas can be applied to smart offices and smart homes.

We do not discuss CV for medical image-related applications in this paper. Although CV has been studied extensively, the techniques used are mostly lower-level image processing. Interested readers can refer to [28].

INTRODUCTION TO VIDEO SURVEILLANCE AND COMPUTER VISION

Traditional video surveillance is labor intensive and usually not very effective. Video surveillance with computer vision techniques, however, saves on labor and provides a consistent monitoring quality. The input to a video surveillance system is video streams from a single or multiple cameras. The system analyzes the video content by separating the foreground from the background, detecting and tracking the objects, and performing a high-level analysis. The high-level analysis provides results such as a scenario being normal or abnormal. The human operator can then focus on the abnormal scenarios and not have to stare at the video trying to find any anomaly.

A general scheme of the video surveillance system is shown in Figure 8, where

- A "Foreground/Background (FG/BG) Detection" module performs FG/BG classification of each image pixel.
- A "Blob Entering Detection" module uses the result (FG mask) of the "FG/BG Detection" module to detect that a new blob object enters the scene.
- A "Blob Tracking" module is initialized by the "Blob Entering Detection" module. This module tracks each blob from the tracked blob list.
- A "Trajectory Generation" module collects all blobs' positions and saves each blob trajectory to hard disk when the motion of the object is no longer presented (for example, when the tracking is lost).
- A "Trajectory Post Processing" module executes a smoothing function on a blob trajectory. This module is optional and need not be included in a specific pipeline.
- A "Trajectory Analysis" module performs a blob trajectory analysis and detection of abnormal trajectories.

We will discuss each module in later sessions.

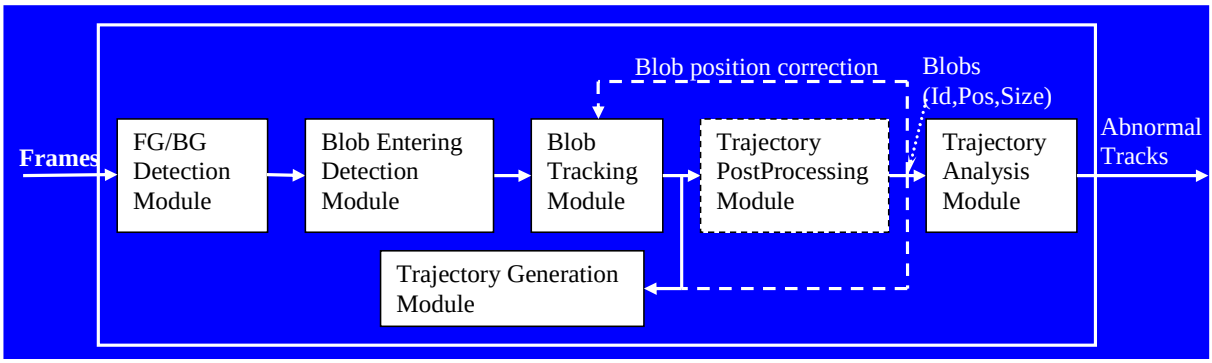


Figure 8: Video surveillance system pipeline

Foreground and Background Estimation

Given a video sequence, extracting the FG from the BG is an important step in the whole video surveillance pipeline. FG estimation is the first stage in the pipeline. Its accuracy affects the accuracy of the later stages. It could affect the performance of the later stages as well. Note that in this paper we use "accuracy" for algorithm accuracy, and "performance" for computation performance such as the speed of computation.

FG detection is generally easier in the indoor environment. Ideally, we want FG/BG estimation to work well in both indoor and outdoor environments. The outdoor environment is more complex, as wavering tree branches, flickering water surfaces, periodic opening and closing of doors, etc. are occurring. We use the approach proposed by Li et al. [29] in our video surveillance system, for its capability in processing complex backgrounds. This method is based on pixel color and color co-occurrence statistics. The pixel color and color co-occurrence distributions are represented by histograms. Bayes decision rule is applied to classify the pixel to BG or FG pixel. The BG is updated after the classification of pixels. The algorithm can successfully handle gradual as well as sudden BG changes, and stationary as well as moving objects.

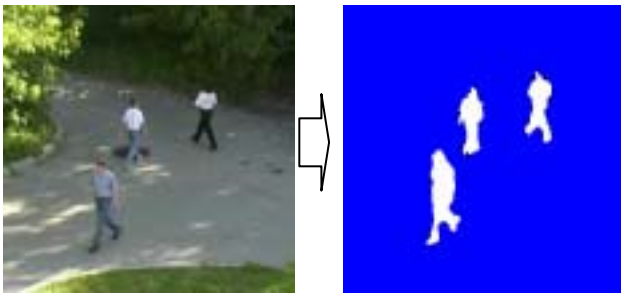


Figure 9: Foreground extraction in outdoor environment

Blob Entering Detection

Our blob detection module is based on a connected component tracker [30]. It does the following:

1. Calculates connected components of the FG mask obtained by the FG/BG estimation module. Each component is considered as a blob.
2. Tracks each blob by trying to find it in the current and the previous frame.
3. Adds a new blob into the tracked blob list if it can be tracked successfully across multiple successive frames.

With the tracked blob list, we can apply object detectors such as those proposed by [31][32] to determine the class of the blob objects: "human," "car," or "unknown."

Blob Tracking

The blob tracking module provides frame-by-frame tracking of the blob position and size. We developed a hybrid object tracker that consists of two components. The first one is a connected-component tracker. It provides reliable and fast tracking results when there is no overlap of two human blobs. The second component is a tracker

that is based on mean-shift algorithms and particle filtering [33][34]. A Kalman filter is used to predict the position of the blob in the next frame, thus implying that overlap will occur in the next frame. If overlap is to occur, the second component, the particle filter-based tracker, is used. Otherwise, the fast connected-component-based tracker is used.

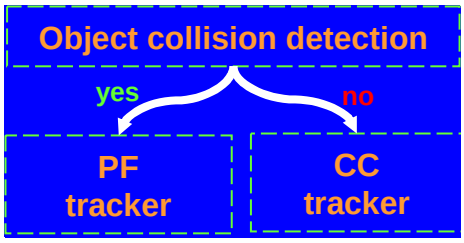


Figure 10: Switching of trackers

Activity Analysis

To detect the anomaly of a scene, we classify the blob trajectories. Trajectory analysis approaches can be found in [35]-[38]. To detect abnormal trajectories, a histogram approach is used. This method treats a trajectory as an independent set of feature vectors. Each feature vector includes such features as blob position, blob velocity, and blob state duration. A 5D histogram of these features is continuously collected and analyzed. Thus, if the current blob has features that were never or rarely observed before, then the blob and its trajectories are classified as abnormal.

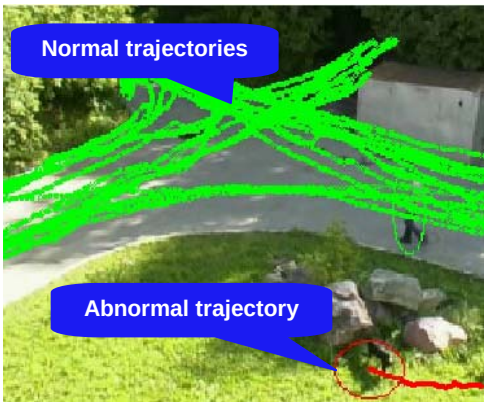


Figure 11: Abnormal trajectory

WORKLOAD ANALYSIS OF THE VIDEO SURVEILLANCE SYSTEM

In this section, we profile the video surveillance system by the Intel® VTune™ Performance Analyzer [39]. We identify hot spots for future performance improvement, either in algorithm modification or in hardware acceleration. Furthermore, we explore the distribution of the operations.

Profiling our system shows that the most computationally expensive modules of the whole video surveillance system are FG detection (up to 95% of execution time) and object tracking (about 5% when only the connected components method is used, and up to 20% when mean shift algorithms with particle filtering is employed). Histogram-based trajectory analysis doesn't take a significant portion of the computational resources; although more sophisticated techniques may potentially contribute more to this workload. Note that the numbers given are not universal to all kinds of scenarios including both indoor and outdoor environments. In a scene where there are more objects presented, object tracking may consume a larger proportion of computational resources, whereas the resources used by FG/BG estimation remain about the same.

The most computationally expensive module of the pipeline, FG/BG estimation, consumes 1 billion microinstructions per frame of size 720x576. On a 3.2 GHz Intel® Pentium® 4 processor, therefore, it takes 0.4 sec.

We further profile the FG/BG module as shown in Figure 12. The most expensive part of the algorithm is the histogram update, which scans all histogram bins. Classification uses only a subset of the histogram. Other parts of the algorithm work only with frame pixels, searching the place for the current pixel value.

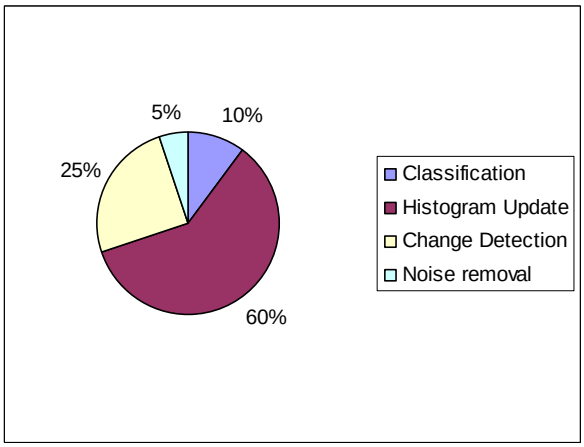


Figure 12: Foreground detection algorithm profile

® Intel and VTune are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States and other countries.

® Intel and Pentium are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States and other countries.

The algorithm consumes a large amount of memory. Each pixel keeps a set of color and color co-occurrence histograms, which takes up about 1 kb per pixel. The whole frame, of size 720x576, needs about 400 MB of memory. However, according to the memory statistics (see Table 1), accesses to the histogram are mostly cached, so we don't have to read the same histogram values from the memory several times; we can get them from the cache.

Table 1: Memory characteristics of foreground detection algorithm

L1 cache hit rate	90%
L2 cache hit rate	77%
Bus data traffic per frame	134 MB
Bus utilization	6%

We now look at the operation distribution (as shown in Figure 13). Most of the arithmetic operations are from integer operations. Operations done on pixel values and histogram bins are mostly integer operations. We see similar situations in many CV workloads where pixel values, histograms, and array indices calculation, are often involved. Floating-point operations in FG/BG estimation are minimal, since the FG/BG algorithm accesses floating-point data only for the histogram bin that is hit by the pixel value. The branches portion is noticeable. However, the branch prediction is good (about 90% correct). Therefore, mis-predicted branches do not significantly impact performance.

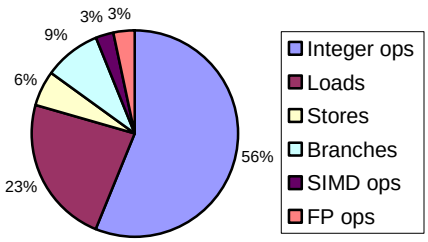


Figure 13: Operations distribution for foreground detection algorithm

SUMMARY

RMS will be the key to future data processing. CV workload is one important RMS workload. In this paper, we talked about trends in CV algorithms and applications. Understanding the trends in CV algorithms and identifying trends in CV applications help Intel in developing future computing platforms.

We then focused on video surveillance systems. A complete video surveillance pipeline captures important aspects of many CV workloads. Video surveillance is one of the most important and resource-demanding CV applications. We identified the hot spots and operation distributions of the system using the Intel VTune Performance Analyzer. Such performance analysis results will be useful for future Intel architecture innovations.

ACKNOWLEDGMENTS

The authors thank Christopher J. Hughes, Yen-Kuang Chen, and Sanjeev Kumar for providing insights into this paper and for editing the technical content. We also thank the paper referees for their valuable comments.

REFERENCES

[1] R. Hammond and R. Mohr, "Mixture densities for video objects recognition," 2000 *IEEE International Conference on Pattern Recognition*, pp. 71-75.

[2] R. Wilson, "MGMM: multiresolution Gaussian mixture models for computer vision," 2000 *IEEE International Conference on Pattern Recognition*, pp. 212-215.

[3] M. Harville, G. Gordon, and J. Woodfill, "Foreground segmentation using adaptive mixture models in color and depth," *2001 IEEE Workshop on Detection and Recognition of Events in Video*, pp. 3-11.

[4] Y. Zhu and K. Fujimura, "Driver face tracking using Gaussian mixture model (GMM)," *IEEE Intelligent Vehicles Symposium*, June 2003, pp. 587-592.

[5] Z. Zivkovic, "Improved adaptive Gaussian mixture model for background subtraction," ICPR 2004, pp. 28-31.

[6] C. Stauffer C, W.E.L. Grimson, "Adaptive background mixture models for real-time tracking," 1999 *IEEE CVPR*.

[7] M.S. Arulampalam, S. Maskell, N. Gordon, and T. Clapp, "A tutorial on particle filters for online nonlinear/non-Gaussian Bayesian tracking," *IEEE Trans. Signal Processing*, 50(2), Feb. 2002, pp. 174-188.

[8] F. Gustafsson, F. Gunnarsson, N. Bergman, U. Forssell, J. Jansson, R. Karlsson, and P.-J. Nordlund, "Particle filter for positioning, navigation, and

tracking," *IEEE Trans. On Signal Processing*, 50(2), Feb. 2002, pp. 425-437.

[9] P.M. Djuric, J.H. Kotecha, J. Zhang, Y. Huang, T. Ghirmai, M.F. Bugallo, and J. Miguez, "Particle filtering," *IEEE Signal Processing Magazine*, Sept. 2003, pp. 19-38.

[10] Z. Tu and S.-C. Zhu, "Image Segmentation by Data-Driven Markov Chain Monte," *IEEE Trans. On PAMI*, 24(5), May 2002, pp. 657-673.

[11] J.C. Spall, "Estimation via Markov Chain Monte Carol," *IEEE Control Systems Magazine*, April 2003, pp. 34-45.

[12] J. Barron, D. Fleet, and S. Beauchemin, "Performance of optical flow techniques," *Int. J. Comput.Vis.*, vol. 12, no. 1, 1994, pp. 42-77.

[13] A. Verri and T. Poggio, "Motion field and optical flow: qualitative properties," *IEEE Trans. On PAMI*, 11(5), May 1999, pp. 490-498.

[14] A. Giachetti, M. Campani, and V. Torre, "The use of optical flow for road navigation", *IEEE Trans. On Robotics and Automation*, 14(1), Feb. 1998, pp. 34-48.

[15] H. Sakoe and S. Chiba, "Dynamic programming optimization spoken word recognition," *IEEE Trans. Acoust., Speech, Signal Processing*, vol. ASSP-26, Feb. 1978, pp. 43-49. (DTW).

[16] D. Berndt and J. Clifford, "Using dynamic time warping to find patterns in time series," *AAAI-94 Workshop on Knowledge Discovery in Databases (KDD-94)*, Seattle, WA, 1994. (DTW).

[17] *IEEE Trans. On PAMI* 22(8), August 2000, Special Section on Video Surveillance, R.T. Collins, A.J. Lipton, and T. Kanade, guest editors.

[18] A. Hampapur, L. Brown, J. Connell, A. Ekin, N. Haas, M. Lu, H. Merkl, S. Pankanti, A. Senior, C.-F. Shu, and Y.L. Tian, "Smart Video Surveillance: exploring the concept of multiscale spatiotemporal tracking," *IEEE Signal Processing Magazine*, March 2005, pp. 38-51.

[19] J. Manigel and W. Leonard, "Vehicle control by computer vision," *IEEE Trans. On Industrial Electronics*, 39(3), June 1992, pp. 181-188.

[20] *IEEE Trans. On Vehicular Technology*, 53(6), Nov. 2004, Special Section on In-Vehicle Computer Vision Systems, R. Cucchiara, D. Lovell, A. Prati, and M.M. Trivedi, guest editors.

[21] GM Collaborative Research Lab at Carnegie Mellon University, http://gm.web.cmu.edu/*

[22] C. Demonceaux, A. Potelle, and D. Kachi-Akkouche, "Obstacle detection in a road scene based on motion analysis," *IEEE Trans. On Vehicular Technology*, 53(6), Nov. 2004, pp. 1649-1656.

[23] M. Bertozzi, A. Broggi, A. Fascioli, T. Graf, and M.-M. Meinecke, "Pedestrian detection for driver assistance using multiresolution infrared vision," *IEEE Trans. On Vehicular Technology*, 53(6), Nov. 2004, pp. 1666-1678.

[24] Photo courtesy: Professor Mohan Trivedi, Laboratory for Intelligent and Safe Automobiles, UCSD, cvrr.ucsd.edu/LISA Ref: M.M. Trivedi, S.Y. Cheng, E.M.C. Childers, and S.J. Krotosky, "Occupant posture analysis with stereo and thermal infrared video: algorithms and experimental evaluation," *IEEE Trans. On Vehicular Technology*, 53(6), Nov. 2004, pp. 1698-1712.

[25] K.N. Kutulakos and J.R. Vallino, "Calibration-free augmented reality," *IEEE Trans. On Visualization and Computer Graphics*, 4(1), Jan.-March 1998, pp. 1-20.

[26] J. Ohya, J. Kurumisawa, R. Nakatsu, K. Ebihara, S. Iwasawa, D. Iwasawa, and T. Horprasert, "Virtual metamorphosis," *IEEE Multimedia*, 6(2), April-June 1999, pp. 29-39.

[27] A. Mihailidia, B. Carmichael, and J. Boger, "The use of computer vision in an intelligent environment to support aging-in-place, safety, and independence in the home," *IEEE Trans. On Information Technology in Biomedicine*, 8(3), Sept. 2004, pp. 238-247.

[28] J.S. Duncan and N. Ayache, "Medical image analysis: progress over two decades and the challenges ahead," *IEEE Trans. On PAMI*, 22(1), Jan. 2000, pp.85-106.

[29] L. Li, W. Huang, I.Y.H. Gu, Q. Tian, "Foreground object detection from videos containing complex background," *ACM Multimedia**, 2003.

[30] A. Senior, A. Hampapur, Y.-L. Tian, L. Brown, S. Pankanti, R. Bolle, "Appearance Models for Occlusion Handling," in *proceedings of Second International workshop on Performance Evaluation of Tracking and Surveillance systems in conjunction with CVPR'01*, December 2001.

[31] P. Viola, M.J. Jones, and D. Snow, "Detecting Pedestrians Using Patterns of Motion and Appearance," *IEEE International Conference on Computer Vision (ICCV)*, vol. 2, pp. 734-741, 2003.

[32] H. Schneiderman and T. Kanade, "A statistical model for 3d object detection applied to faces and cars," *IEEE Conf. Computer Vision and Pattern Recognition (CVPR)*, June 2000.

[33] D. Comaniciu, V. Ramesh, P. Meer, "Real-time tracking of non-rigid objects using mean shift," *IEEE International Conference on Pattern Recognition*, vol. 2, pp. 142-149, 13-15 June, 2000.

[34] K. Nummiaro, E. Koller-Meier, and L. Van Gool, "A color based particle filter," in *First International Workshop on Generative-Model-Based Vision*, A.E.C. Pece, Ed., 2002.

[35] N. Johnson and D.C. Hogg, "Learning the distribution of object trajectories for event recognition," *Image and Vision Computing*, 14:609-615, 1996.

[36] N. Sumpter., A.J. Bulpitt, "Learning spatio-temporal patterns for predicting object behaviour," *Image and Vision Computing, 18(9)*, pages 697-704, 2000.

[37] J. Owens, A. Hunter, "Application of the Self-Organising Map to Trajectory Classification," Proc. Third IEEE Visual Surveillance Workshop, 1 July 2000, Dublin, ISBN 0-7695-0698-4, pp. 77-83.

[38] J. Lou, Q. Liu, and T. Tan, W. Hu, "Semantic interpretation of object activities in a surveillance system," *Pattern Recognition, 2002*, Volume 3, 11-15 Aug. 2002, pp. 777-780.

[39] Intel VTune Performance Analyzers. http://www.intel.com/software/products/vtune/.

AUTHORS' BIOGRAPHIES

Trista P. Chen is a staff research scientist at Intel's Application Research Lab. She received her Ph.D. degree in Electrical and Computer Engineering from Carnegie Mellon University and M.S. and B.S. degrees from National Tsing Hua University, Taiwan. Her research interests include bridging the gap between computer vision and computer graphics, microprocessor architecture, and multimedia communications. Prior to joining Intel, she was with Nvidia, architecting the first video processor for Nvidia graphic processor units. She was with Sony Design Center and HP Cambridge Research Center in the summers of 2000 and 2001, respectively. Her e-mail is trista.p.chen at intel.com.

Alexander Bovyrin is a senior research scientist at Intel's Application Research Lab. He received his M.S. degree in Applied Mathematics and Cybernetics from Nizhny Novgorod State University in 1999 and his Ph.D. degree in the Theory of Dynamic System Control from Nizhiny Novgorod State University of Civil Engineering in 2003. Before joining Intel Corp. in 2000, he worked on computer vision industrial applications. He is interested in computer vision and machine learning. His e-mail is alexander.bovyrin at intel.com.

Roman Belenov is a research scientist in Intel's Application Research Lab. His research interests include video compression and processing, multimedia software architecture, microprocessor architecture, and wireless networking. He received a Diploma in Physics from Nizhny Novgorod State University. His e-mail is roman.belenov at intel.com.

Konstantin Rodyushkin is a senior research scientist at Intel's Application Research Lab. He received his Ph.D degree in 2002 from Nizhny Novgorod State University. His research interests are in computer vision and machine learning. His e-mail is Konstantin.Rodyushkin at intel.com.

Alexander Kuranov has been a software engineer at the Intel Russia Research Center since 2001. He graduated from Nizhny Novgorod State University in 2002. His main interests are computer vision and data-mining algorithms. His e-mail is Alexander.Kuranov at intel.com.

Victor Eruhimov is a senior research scientist in the Intel Russia Research Center. He is leading a team of researches investigating the computational properties of algorithms in the area of computer vision and machine learning. He was a senior researcher in the Open Computer Vision project, working on the development of the OpenCV library and conducting research in computer vision. Victor received a Masters degree from the Advanced School of General and Applied Physics in the Institute of Applied Physics, Russian Academy of Sciences, in 1999. His e-mail is victor.eruhimov at intel.com.

Horst Haussecker is a principal engineer in Intel's Corporate Technology Group and manager of the Computational Nano-Vision research project. His research interests include model-based computer vision and application of digital image processing as a quantitative instrument. Horst is co-editor and author of two textbooks in Computer Vision, and he has authored more than 50 peer-reviewed technical articles. He received his M.S. and Ph.D. degrees in Physics from Heidelberg University, and prior to joining Intel in 2001, he was a researcher at the Xerox Palo Alto Research Center (PARC), where he was involved in image sequence analysis and information processing in sensor networks. His e-mail is horst.haussecker at intel.com.

Learning-Based Computer Vision with Intel's Open Source Computer Vision Library

Gary Bradski, Corporate Technology Group, Intel Corporation
Adrian Kaehler, Enterprise Platforms Group, Intel Corporation
Vadim Pisarevsky, Software and Solutions Group, Intel Corporation

Index words: computer vision, face recognition, road recognition, optimization, open source, OpenCV

ABSTRACT

The rapid expansion of computer processing power combined with the rapid development of digital camera capability has resulted in equally rapid advances in computer vision capability and use. Intel has long been at the forefront of enabling this advance on the computer hardware and software side. Computer vision software is supported by the free Open Source Computer Vision Library (OpenCV) that optionally may be highly optimized by loading the commercial Intel Integrated Performance Primitives (IPP). IPP now automatically supports OpenCV with no need to change or even recompile the user's source code. This functionality enables development groups to deploy vision and provides basic infrastructure to experts in vision.

OpenCV has long supported "geometric vision" from camera calibration, motion tracking in 2D, finding the camera location given a known 3D object, on up to producing depth maps from stereo vision. This paper describes using OpenCV for "learning-based vision," where objects such as faces, or patterns such as roads, are learned for segmentation and recognition.

INTRODUCTION

The Open Source Computer Vision Library (OpenCV) [1] is a free, open source collection of computer vision routines geared mainly towards human-computer interaction, robotics, security, and other vision applications where the lighting and context of use cannot be controlled. OpenCV is not geared towards factory machine vision where the environmental conditions can be controlled and one generally knows what one is looking for, although there is a large overlap.

OpenCV was designed for enablement and infrastructure. Many groups who could make use of vision were prevented from doing so due to lack of expertise; OpenCV enables these types of groups to add functionality such as face finding and tracking in a few lines of code. Other groups have vision expertise but were uselessly recreating vision algorithms that were already standard; OpenCV provides experts with a solid vision infrastructure and thereby allows experts to work at a higher level rather than have to worry about the basics. Because of the above, OpenCV's BSD type license is designed to promote free commercial and research use. Optionally, users may install the IPP libraries and benefit from highly optimized code without needing to recompile via the use of automatically selected optimized dynamic linked libraries.

OpenCV support for vision is extensive. It supports routines for input, display, and storage of movies and single images. Image-processing debug is supported by drawing and text display functions. Image processing itself is handled through convolution, thresholding, morphological operations, floodfills, histogramming, smoothing, pyramidal sub-sampling and a full suite of image algebra and arithmetic. Geometry is supported by Delaney triangulation, calibration, fundamental and essential matrices computation, image alignment, and stereo depth calculation. A full range of feature detection algorithms exists from corner detectors, Canny edge operators, blob finders, scale invariant features, and so on. Shape descriptors such as Hu moments, contour processing, Fourier descriptors, convex hulls, and connected components exist. Motion is covered via several types of optical flow, background learning and differencing, motion templates, and motion gradients. Learning-based vision is supported through feature histogram comparison, image statistics, template-based correlation, decision trees, and statistical boosting on up to convolutional neural networks.

OpenCV was released in Alpha in 2000, Beta in 2003, and will be released in official version 1.0 in Q4 2005. If the Intel Integrated Performance Primitives (IPP) library [2] is

optionally installed, OpenCV will automatically take advantage of and swap in the hand optimized routines in IPP providing a substantial speed-up to many vision routines.

In this paper, we describe computer vision routines based on learning. Learning-based vision has applications to image-based Web mining, image retrieval, video indexing, security, etc. OpenCV has strong and growing support for learning-based vision. We start out, however, by first discussing a recent change to OpenCV, full IPP support, and then move on to discuss two learning applications. We begin by describing automatic optimization of OpenCV using IPP and then we discuss using OpenCV for learned object finding and tracking (face), and end with abstract pattern segmentation (road finding).

AUTOMATIC OPTIMIZATION USING INTEGRATED PERFORMANCE PRIMITIVES

How to Make Use of IPP

Intel Integrated Performance Primitives (IPP) library is a large collection of low-level computational kernels highly optimized for Intel architectures, including the latest Pentium®, Itanium®, and XScale® processors. It consists of multiple domains that reside in separate dynamic libraries: signal and image processing, matrix processing, audio and video codecs, computer vision, speech recognition, cryptography, data compression, text processing, etc. It can be retrieved from http://www.intel.com/software/products/ipp [2]; full evaluation versions for Windows* and Linux*, and a free non-commercial version for Linux are available.

OpenCV is able to automatically detect and use the IPP library once the latter is installed; there is no need to recompile it. On Windows, make sure that the bin subdirectory of IPP is in the system path, for example, if IPP is installed to `"C:\Program Files\Intel\IPP"`, add `"C:\Program Files\Intel\IPP\bin"` to the path. On Linux the IPP dynamic libraries should be already in one of the standard paths after installation.

To check whether OpenCV has found IPP or not, the user application may call the `cvGetModuleInfo()` function:

```
const char* opencv_libraries = 0;
const char* addon_modules = 0;
cvGetModuleInfo( 0, &opencv_libraries,
                 &addon_modules );
printf( "OpenCV: %s\nAdd-on Modules: %s\n",
        opencv_libraries, addon_modules );
```

When IPP is detected, it will print something like this:

```
OpenCV: cxcore: beta 4.1 (0.9.7), cv: beta 4.1 (0.9.7)
Add-on modules: ippcv20.dll, ippi20.dll, ipps20.dll, ippvm20.dll
```

where ipp*.dll are names of IPP components, used by OpenCV: ippcv – computer vision, ippi – image processing, ipps – signal processing, ippvm – fast math functions.

Note that the functions in ippi20.dll and the other '20' libraries do not contain the processing functions themselves. They are proxies for CPU-specific libraries (ippia6.dll, ippiw7.dll etc. for IPPI) that are loaded by the IPP dispatcher. The dispatcher mechanism and other concepts behind IPP are explained in detail in the IPP book [3].

How Automatic Use of Optimized IPP Works

The mechanism to swap in optimized code if found is simple. It uses function pointers and dynamic library-loading facilities provided by the operating system (OS). For every IPP function that OpenCV can use there is a pointer to a function that is initially set to null and which is assigned to a valid address when the corresponding IPP component is detected and loaded. So, while OpenCV can benefit from using IPP, it does not depend on it; the functionality is the same, regardless of whether IPP is installed or not. That is, for every IPP function there is a backup C code that is always included inside OpenCV binaries. So, a higher-level external OpenCV function loads a function pointer that calls either optimized IPP code or embedded low-level OpenCV C code depending on which is available.

Let's consider an example. The function cvPyrDown reduces image size down one level by employing Gaussian smoothing and sub-sampling. Smoothing is preformed prior to sub-sampling so that spurious frequencies are not introduced due to violations of the Nyquist sampling theorem. cvPyrDown supports multiple image types via several lower-level functions. In particular, 8-bit single-channel images are processed with icvPyrDownG5x5_8u_CnR. The corresponding IPP function for this type of images is ippiPyrDown_Gauss5x5_8u_C1R.

® Pentium, Itanium, and XScale are all registered trademarks of Intel Corporation or its subsidiaries in the United States and other countries.

* All other brands and names are the property of their respective owners.

So we have the following code (simplified compared to the actual implementation):

```
// --- cvpyramids.cpp: ---
// declaration of the function type.
// Matches to the declaration in IPP header
files
typedef int (CV_STDCALL *
icvPyrDown_Gauss5x5_8u_C1R_t)( const uchar* src,
int src_step, uchar* dst, int dst_step, CvSize
size, void* buffer );
// pointer to the IPP function
icvPyrDown_Gauss5x5_8u_C1R_t
       icvPyrDown_Gauss5x5_8u_C1R_p = 0;


// C implementation, embedded in OpenCV
static int icvPyrDownG5x5_8u_CnR(
    const uchar* src, int src_step,
    uchar* dst, int dst_step, CvSize size,
    void* buffer, int cn )
{
   …
   return CV_OK;
}


// external high-level function
void cvPyrDown( const CvArr* src_arr,
                CvArr* dst_arr,
                int filter )
{
    …
    if( data_type == CV_8UC1 ) {
        if( icvPyrDown_Gauss5x5_8u_C1R_p )
            icvPyrDown_Gauss5x5_8u_C1R_p(…);
        else
            icvPyrDownG5x5_8u_CnR(…,1);
    }
    …
}
```

Also, the function pointer and the related information are stored to the joint table that is used by the OpenCV initialization procedure (a.k.a. switcher).

//cvswitcher.cpp:

…

```
{ (void**)&icvPyrDown_Gauss5x5_8u_C1R_p, 0,
 "ippiPyrDown_Gauss5x5_8u_C1R",
CV_PLUGINS1(CV_PLUGIN_IPPI), 0 },
```

…

That is, each entry of the table contains a pointer to the function pointer (so that the address could be changed by the switcher), the real function name, and the id of the module that contains the function (IPPI ~ "ippi20.dll" in this case). On start-up, the OpenCV initialization procedure tries to locate and load IPP modules:

…

```
plugins[CV_PLUGIN_IPPI].handle =
    LoadLibrary("ippi20.dll");
```

and retrieve the function pointers:

```
for(…;…;…) {
   void* handle =
plugins[func_table[i].plugin_id].handle;
   const char* fname=func_table[i].func_names;
   if( handle )
      *func_table[i].func_addr =
           GetProcAddress( handle, fname );
}
```

(on Linux `dlopen` is used instead of `LoadLibrary` and `dlsym` instead of `GetProcAddress`).

Functionality Coverage

Currently, OpenCV knows of and can use over 300 IPP functions. Below is a table of the major functions and the approximate speed-up (on a Pentium 4 processor) that a user could get by on using IPP. Note that the wide range of speed-up numbers in the table results from different potential image types. Byte images are faster to process than integer images which are faster than floating, for example. Another timing difference results from different kernel sizes. Small kernels are faster than large kernels, and some common kernels are "hard wired"–hand optimized.

Table 1: Approximate speed-ups using assembly optimized IPP over the embedded optimized C in OpenCV

Function	Speed-up range (OpenCV/IPP exec. time)
Gaussian Pyramids	~3
Morphology	~3-7
Median filter	~2.1-18
Linear convolution (with a small kernel)	~2-8
Template Matching	~1.5-4
Color Conversion (RGB to/from Grayscale, HSV, Luv)	~1-3
Image moments	~1.5-3
Distance transform	~1.5-2
Image affine and perspective transformations	~1-4
Corner detection	~1.8
DFT/FFT/DCT	~1.5-3
Math functions (exp, log, sin, cos …)	3-10

In OpenCV 1.0, support for more IPP functions, such as face detection and optical flow, will be added.

FACE DETECTION

Introduction/Theory

Object detection, and in particular, face detection is an important element of various computer vision areas, such as image retrieval, shot detection, video surveillance, etc. The goal is to find an object of a pre-defined class in a static image or video frame. Sometimes this task can be accomplished by extracting certain image features, such as edges, color regions, textures, contours, etc. and then using some heuristics to find configurations and/or combinations of those features specific to the object of interest. But for complex objects, such as human faces, it is hard to find features and heuristics that will handle the huge variety of instances of the object class (e.g., faces may be slightly rotated in all three directions; some people wear glasses; some have moustaches or beards; often one half of the face is in the light and the other is shadow, etc.). For such objects, a statistical model (classifier) may be trained instead and then used to detect the objects.

Statistical model-based training takes multiple instances of the object class of interest, or "positive" samples, and multiple "negative" samples, i.e., images that do not contain objects of interest. Positive and negative samples together make a training set. During training, different features are extracted from the training samples and distinctive features that can be used to classify the object are selected. This information is "compressed" into the statistical model parameters. If the trained classifier does not detect an object (misses the object) or mistakenly detects the absent object (i.e., gives a false alarm), it is easy to make an adjustment by adding the corresponding positive or negative samples to the training set.

OpenCV uses such a statistical approach for object detection, an approach originally developed by Viola and Jones [4] and then analyzed and extended by Lienhart [5, 6]. This method uses simple Haar-like features (so called because they are computed similar to the coefficients in Haar wavelet transforms) and a cascade of boosted tree classifiers as a statistical model. In [4] and in OpenCV this method is tuned and primarily used for face detection. Therefore, we discuss face detection below, but a classifier for an arbitrary object class can be trained and used in exactly the same way.

The classifier is trained on images of fixed size (Viola uses 24x24 training images for face detection), and detection is done by sliding a search window of that size through the image and checking whether an image region at a certain location "looks like a face" or not. To detect faces of different size it is possible to scale the image, but the classifier has the ability to "scale" as well.

Fundamental to the whole approach are Haar-like features and a large set of very simple "weak" classifiers that use a single feature to classify the image region as face or non-face.

Each feature is described by the template (shape of the feature), its coordinate relative to the search window origin and the size (scale factor) of the feature. In [3], eight different templates were used, and in [5, 6] the set was extended to 14 templates, as shown in Figure 1.

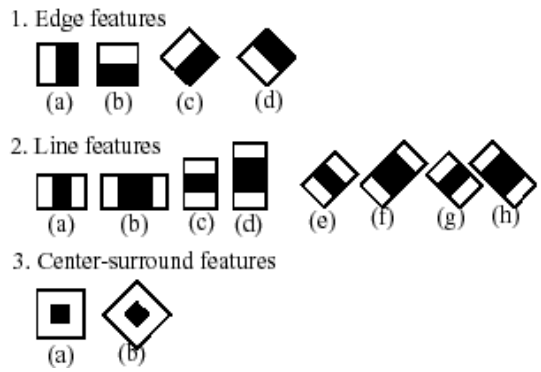


Figure 1: Extended set of Haar-like features

Each feature consists of two or three joined "black" and "white" rectangles, either up-right or rotated by 45°. The Haar feature's value is calculated as a weighted sum of two components: The pixel sum over the black rectangle and the sum over the whole feature area (all black and white rectangles). The weights of these two components are of opposite signs and for normalization, their absolute values are inversely proportional to the areas: for example, the black feature 3(a) in Figure 1 has $\text{weight}_{\text{black}} = -9 \times \text{weight}_{\text{whole}}$.

In real classifiers, hundreds of features are used, so direct computation of pixel sums over multiple small rectangles would make the detection very slow. But Viola [4] introduced an elegant method to compute the sums very fast. First, an integral image, Summed Area Table (*SAT*), is computed over the whole image *I*, where

$$SAT(X,Y) = \sum_{x<X,\, y<Y} I(x,y).$$

The pixel sum over a rectangle $r=\{(x,y), x_0 \le x < x_0+w, y_0 \le y < y_0+h\}$ can then be computed using *SAT* by using just the corners of the rectangle regardless of size:

$$RecSum(r) = SAT(x_0+w, y_0+h) - SAT(x_0+w, y_0) - SAT(x_0, y_0+h) + SAT(x_0, y_0)$$

This is for up-right rectangles. For rotated rectangles, a separate "rotated" integral image must be used.

The computed feature value $x_i = w_{i,0} RecSum(r_{i,0}) + w_{i,1} RecSum(r_{i,1})$ is then used as input to a very simple decision tree classifier that usually has just two terminal nodes, that is:

$$f_i = \begin{cases} +1, & x_i \ge t_i \\ -1, & x_i < t_i \end{cases}$$

or three terminal nodes:

$$f_i = \begin{cases} +1, & t_{i,0} \le x_i < t_{i,1} \\ -1 & else \end{cases}$$

where the response +1 means the face, and –1 – means the non-face. Every such classifier, called a weak classifier, is not able to detect a face; rather, it reacts to some simple feature in the image that may relate to the face. For example, in many face images eyes are darker than the surrounding regions, and so feature 3a in Figure 1, centered at one of the eyes and properly scaled, will likely give a large response (assuming that $weight_{black}<0$).

In the next step, a complex and robust classifier is built out of multiple weak classifiers using a procedure called boosting, introduced by Freund and Schapire [7].

The boosted classifier is built iteratively as a weighted sum of weak classifiers:

$$F = sign(c_1 f_1 + c_2 f_2 + \ldots + c_n f_n)$$

On each iteration, a new weak classifier f_i is trained and added to the sum. The smaller the error f_i gives on the training set, the larger is the coefficient c_i that is assigned to it. The weight of all the training samples is then updated, so that on the next iteration the role of those samples that are misclassified by the already built *F* are emphasized. It is proven in [7] that if f_i is even slightly more selective than just a random guess, then *F* can achieve an arbitrarily high (<1) hit rate and an arbitrarily small (>0) false alarm rate, if the number of weak classifiers in the sum (ensemble) is large enough. However, in practice, that would require a very large training set as well as a very large number of weak classifiers, resulting in a slow processing speed.

Instead, Viola [4] suggests building several boosted classifiers F_k with constantly increasing complexity and chaining them into a cascade with the simpler classifiers going first. During the detection stage, the current search window is analyzed subsequently by each of the F_k classifiers that may reject it or let it go through, as depicted in Figure 2.

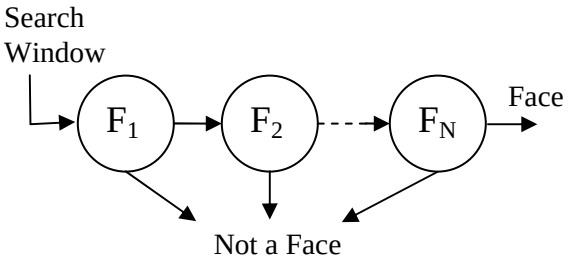


Figure 1: Object (face) detection cascade of classifiers where rejection can happen at any stage

That is, F_k (k=1..N)'s are subsequently applied to the face candidate until it gets rejected by one of them or until it passes them all. In experiments, about 70-80% of candidates are rejected in the first two stages that use the simplest features (about 10 weak classifiers each), so this technique speeds up detection greatly. Most of the detection time, therefore, is spent on real faces. Another advantage is that each of the stages need not be perfect; in fact, the stages are usually biased toward higher hit-rates rather than towards small false-alarm rates. By choosing the desired hit-rate and false-alarm rate at every stage and by choosing the number of stages accurately, it is possible to achieve very good detection performance. For example, if each of the stages gives a 0.999 hit-rate and a 0.5 false-alarm rate, then by stacking 20 stages into a cascade, we will be able to get a hit-rate of 0.999^{20}=0.98 and a false-alarm rate of 0.5^{20}~10^{-6}!

Face Detection with OpenCV

OpenCV provides low-level and high-level APIs for face/object detection. A low-level API allows users to check an individual location within the image by using the classifier cascade to find whether it contains a face or not. Helper functions calculate integral images and scale the cascade to a different face size (by scaling the coordinates of all rectangles of Haar-like features) etc. Alternatively, the higher-level function `cvDetectObjects` does this all automatically, and it is enough in most cases. Below is a sample of how to use this function to detect faces in a specified image:

```
// usage: facedetect --cascade=<path> image_name
#include "cv.h"
#include "highgui.h"
#include <string.h>

int main( int argc, char** argv )
{
    CvHaarClassifierCascade* cascade;

    // face sequence will reside in the storage
    CvMemStorage* storage=cvCreateMemStorage(0);
    IplImage *image;
    CvSeq* faces;
    int optlen = strlen("--cascade=");
    int i;

    if( argc != 3 ||
        strncmp(argv[1], "--cascade=", optlen) )
        return -1;

    // load classifier cascade from XML file
    cascade = (CvHaarClassifierCascade*)
        cvLoad( argv[1] + optlen );
    // load image from the specified file
    image = cvLoadImage( argv[2], 1 );

    if( !cascade || !image )
        return -1;


// get the sequence of face rectangles
faces = cvHaarDetectObjects( image,
          cascade, storage,
          1.2, // scale the cascade
               // by 20% after each pass
          2, // groups of 3 (2+1) or more
neighbor face rectangles are joined into a
single "face", smaller groups are rejected
          CV_HAAR_DO_CANNY_PRUNING, // use Canny
edge detector to reduce number of false alarms
          cvSize(0, 0) // start from the minimum
face size allowed by the particular classifier
    );

    // for each face draw the bounding rectangle
    for(i=0;i<(faces ? faces->total:0); i++ ) {
        CvRect* r = (CvRect*)
             cvGetSeqElem( faces, i );
        CvPoint pt1 = { r->x, r->y };
        CvPoint pt2 = { r->x + r->width,
                        r->y + r->height };
        cvRectangle( image, pt1, pt2,
                     CV_RGB(255,0,0), 3, 8, 0 );
    }

    // create window and show the image with
outlined faces
    cvNamedWindow( "faces", 1 );
    cvShowImage( "faces", image );
    cvWaitKey();
    // after a key pressed, release data
    cvReleaseImage( &image );
    cvReleaseHaarClassifierCascade( &cascade );
    cvReleaseMemStorage( &storage );
    return 0;
}
```

If the above program is built as facedetect.exe, it may be invoked as (type it in a single line):

```
facedetect.exe –-cascade="c:\program
files\opencv\data\haarcascades\haarcascade_
frontalface_default.xml" "c:\program
files\opencv\samples\c\lena.jpg"
```

assuming that OpenCV is installed in c:\program files\opencv. Figure 3 shows example results of using a trained face detection model that ships with OpenCV.

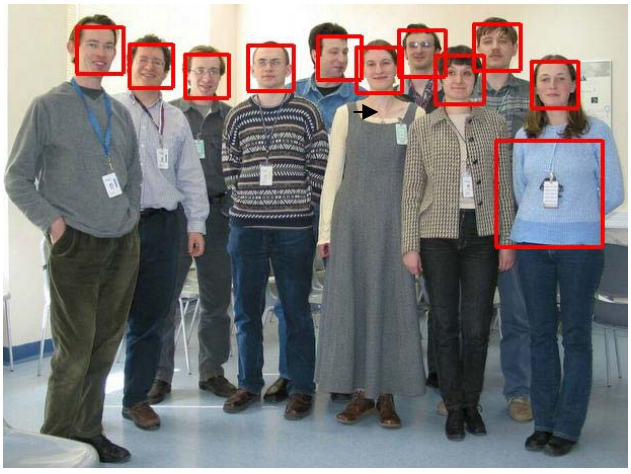


Figure 3: Results of a trained face detection model that ships with OpenCV. All faces were detected, one false positive detection resulted.

A detailed description of object detection functions can be found in the OpenCV reference manual (opencvref_cv.htm, *Object Detection* section).

Training the Classifier Cascade

Once there is a trained classifier cascade stored in an XML file, it can be easily loaded using the `cvLoad` function and then used by `cvHaarDetectObjects` or by low-level object detection functions. The question remains as to how to create such a classifier, if/when the standard cascades shipped with OpenCV fail on some images or one wants to detect some different object classes, like eyes, cars, etc. OpenCV includes a `haartraining` application that creates a classifier given a training set of positive and negative samples. The usage scheme is the following (for more details, refer to the haartraining reference manual supplied with OpenCV):

1. Collect a database of positive samples. Put them into one or more directories and create an index file that has the following format:

   filename_1 count_1 x11 y11 w11 h11 x12 y12 …

   filename_2 count_2 x21 y21 w21 h21 x22 y22 …

   …

   That is, each line starts with a file name (including subdirectories) of an image followed by the number of objects in it and bounding rectangles for every object (x and y coordinates of top-left corner, width and height in pixels). For example, if a database of eyes resides in a directory eyes_base, the index file eyes.idx may look like this:

   ```
   eyes_base/eye_000.jpg  2 30 100  15  10 55 100 15 10
   eyes_base/eye_001.jpg 4 15 20 10 6 30 20 10 6 …
   …
   ```

   Notice that the performance of a trained classifier strongly depends on the quality of the database used. For example, for face detection, faces need to be aligned so that the relative locations of eyes–the most distinctive features–are the same. The eyes need to be on the same horizontal level (i.e., faces are properly rotated) etc. Another example is the detection of profile faces. These are non-symmetric, and it is reasonable to train the classifier only on right profiles (so that variance inside the object class is smaller) and at the detection stage to run it twice–once on the original images and a second time on the flipped images.

2. Build a vec-file out of the positive samples using the `createsamples` utility. While the training procedure might be repeated many times with different parameters, the same vec-file may be re-used.

   Example:

   ```
   createsamples –vec eyes.vec \
   –info eyes.idx –w 20 –h 15
   ```

   The above builds eyes.vec out of the database, described in eyes.idx (see above): all the positive samples are extracted from images, normalized and resized to the same size (20x15 in this case). `createsamples` can also creates a vec file out of a single positive sample (e.g., some company logo) by applying different geometrical transformations, adding noise, altering colors, etc. See *haartraining* in the OpenCV reference html manual for details.

3. Collect a database of negative samples. Make sure the database does not contain instances of the object class of interest. You can make negative samples out of arbitrary images, for example. They can be downloaded from the Internet, bought on CD, or shot by your digital camera. Put the images into one or more directories, and make an index file: that is, a plain list of image filenames, one per line. For example, an image index file called "backgrounds.idx" might contain:

   ```
   backgrounds/img0001.jpg
   backgrounds/my_img_02.bmp
   backgrounds/the_sea_picture.jpg
   …
   ```

4. Run haartraining. Below is an example (type it in command-line prompt as a single line or create a batch file):

   ```
   haartraining
   –data eyes_classifier_take_1
   -vec eyes.vec –w 20 –h 15
   -bg backgrounds.idx
   -nstages 15
   -nsplits 1
   [-nonsym]
   -minhitrate 0.995
   -maxfalsealarm 0.5
   ```

In this example, a classifier will be stored in eyes_classifier_take1.xml. eyes.vec is used as a set of positive samples (of size 20x15), and random images from background.idx are used as negative samples. The cascade will consist of 15 (-nstages) stages; every stage is trained

to have the specified hit-rate (-minhitrate) or higher, and a false-alarm rate (-maxfalsealarm) or lower. Every weak classifier will have just 1 (-nsplits) non-terminal node (1 split trees are called "stumps").

The training procedure may take several hours to complete even on a fast machine. The main reason is that there are quite a lot of different Haar features within the search window that need to be tried. However, this is essentially a parallel algorithm and it can benefit (and does benefit) from SMP-aware implementations. Haartraining supports OpenMP via the Intel Compiler and this parallel version is shipped with OpenCV.

We discussed use of an object detection/recognition algorithm built into OpenCV. In the next section, we discuss using OpenCV functions to recognize abstract objects such as roads.

ROAD SEGMENTATION

The Intel OpenCV library has been used for the vision system of an autonomous robot. This robot is built from a commercial off-road vehicle and the vision system is used to detect and follow roads. In this system, the problem was to use a close-by road, identified by scanning laser range finders to initialize vision algorithms that can extend the initial road segmentation out as far as possible. The roads in question were not limited to marked and paved streets; they were typically rocky trails, fire roads, and other poor quality dirt trails. Figure 4 shows "easy" and "hard" roads.

Figure 4: Example roads: a less difficult power-line access road (top) and a more difficult "off-road" trail (bottom)

Based on laser scanner point clouds in the near field, it was possible to estimate what sections of nearby visible terrain might be road. In this case, "near" is approximately ten meters (varying by terrain type and other ambient conditions). Once projected into the visual camera images, this region could then be used to train a classifier that would extrapolate out beyond the range of the lasers into the far visual field. In many cases the method is successful at extrapolating the road all of the way to the horizon. This amounts to a practical range of as much as one hundred meters. The ability to see into the visual far field is crucial for path planning for high-speed operation.

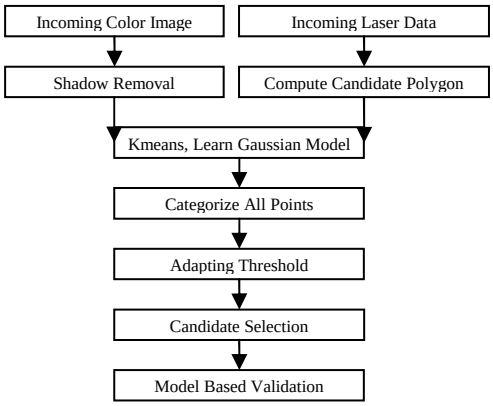


Figure 5: Overview of data flow

The core algorithm outlined in Figure 5 is as follows. First, the flat terrain that the lasers find immediately in front of the vehicle is converted to a single polygon and projected into the camera co-ordinates. Shadow regions are marked out of consideration as shown in Figure 6. The projected polygon represents our best starting guess for determining what pixels in the overall image contribute to road. It is of course possible that there is no road at all. The method is to extrapolate out to find the largest patch to which we can extend what the lasers have given us, and only thereafter to ask if that patch might be a road. This final determination will be made based on the shape of the area found. Only a relatively small set of shapes can correspond to a physical road of approximately constant width, disappearing into the distance.

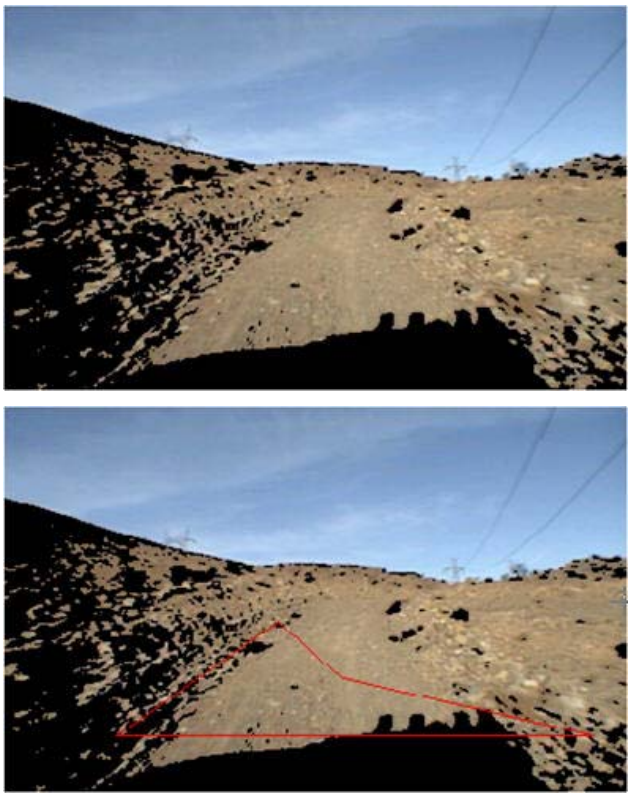


Figure 6: Starting with the bottom image of Figure 4, shadows are eliminated–blacked out– (top), and the polygon is projected from the laser data (bottom)

The red, green, blue (RGB) values of the individual pixels in the marked polygon are used as features to describe individual pixels in the marked region (giving a total of three dimensions). These pixels are kept in a FIFO circular buffer of sufficient depth to hold pixels from many frames. At every frame the OpenCV function cvKMeans2 is called with settings to find three clusters. These clusters are modeled with three multivariate Gaussian models trained from the pixel density distribution in the 3-dimensional color space. This algorithm achieves the same results as fitting three multivariate Gaussians using expectation maximization, but is faster and takes advantage of the built in cvKMeans2 in OpenCV. Three clusters are used because empirically it was found that ruts and rocks in the road tend to give dirt roads a tri-tonal color profile.

After generating the above model, we then use it to score the remaining pixels in the scene by using the OpenCV function cvMahalanobis to find the minimum Mahalanobis distance R_m from the candidate pixel to the means of the three learned multi-modal Gaussian distributions. Pixels for which R_m is less than 1 are assigned a score of 1.0; those for which R_m is greater than 1 are given a score of $1/R_m$. The resulting "image" contains a kind of confidence score for every pixel being road or not. This scoring is essentially equivalent to a log-likelihood score for a pixel being in the road distribution, but again is fast and convenient to compute using cvMahalanobis.

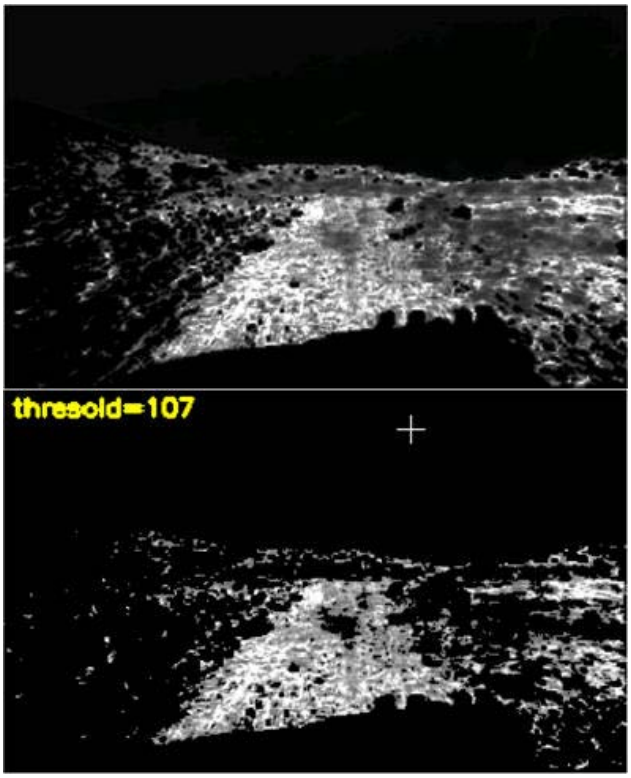


Figure 7: Probability map based on individual pixel's distance from the model mean (top), and image with low probability pixels removed (bottom). In the thresholded bottom image, the threshold is set so as to keep constant the proportion of non-zero pixels in the original

This method does not guarantee that all pixels in the polygon training area will be scored as road. OpenCV function cvThreshold is used adaptively such that no less than 80% of the pixels in the training area are retained after the threshold is applied, as shown in Figure 7. A morphological function cvDilate is then applied with the result that the identified pixels can be clustered into large connected regions. As we are only interested in extrapolating the already identified space found by the lasers, clusters are next rejected if they do not connect with the polygonal training area cluster. The retained clusters are then used to mask the original confidence image. This process is illustrated in Figure 8.

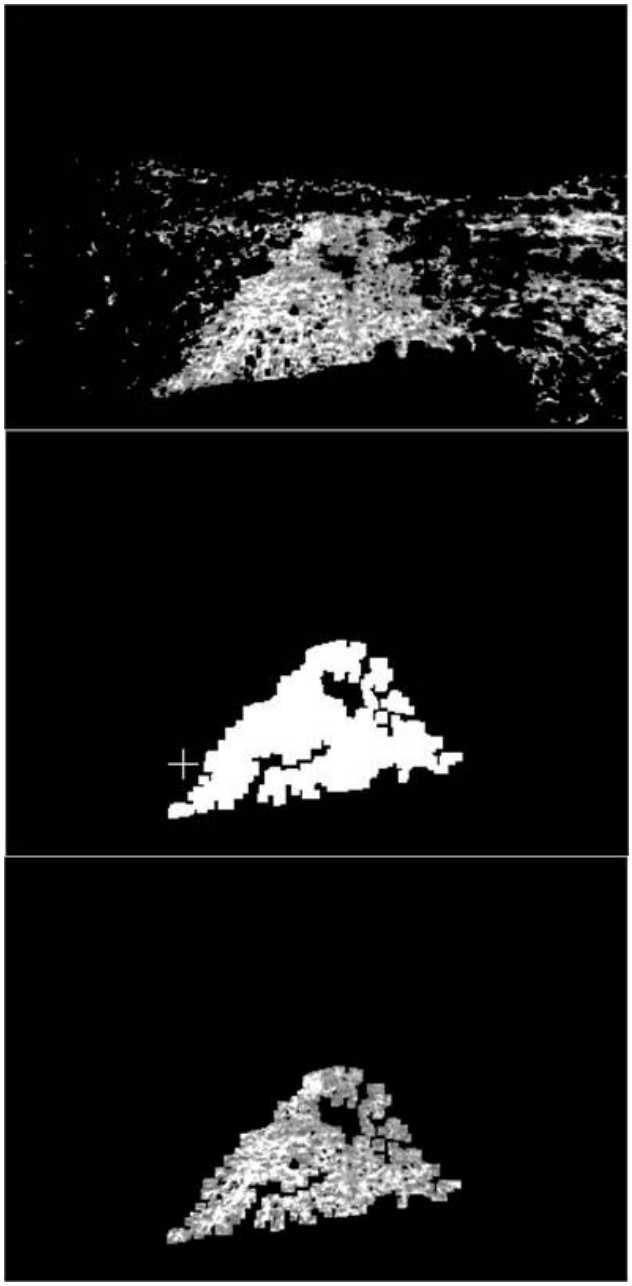


Figure 8: Starting with the thresholded image (top), the contiguous patch that intersects the training polygon is identified (middle) and clipped from the original probability map (bottom)

Finally, in Figure 9, the resulting clusters are compared with the expected shape of a road projected into the camera coordinates. This is done by fitting the right and left sides of the cluster(s) to individual lines, and by computing an effective horizon which is the location of the highest identified road pixel. If these fits are sufficiently good, the fit lines intersect at the horizon, and the angles of the lines are consistent with the projection of a road into the camera coordinates, then the system reports the confidence image. This image, once projected into ground-based coordinates, is used to mark the road ahead for the subsequent path-planning operations.

By building this model, it is possible to reject anomalies that might arise from dirt, facing into the sun, reflections, and other sources. Such anomalies are not uncommon, and the ability to build a geometrically possible model and then score that model allows the system to report correctly when the found pixels are something other than a road. In some cases this occurs when no road is present at all, and the lasers find only a smooth patch of ground that is not part of an actual road. In three hours of test video varying widely over many terrain types containing both road and no road, the algorithm was not seen to give a false positive signal.

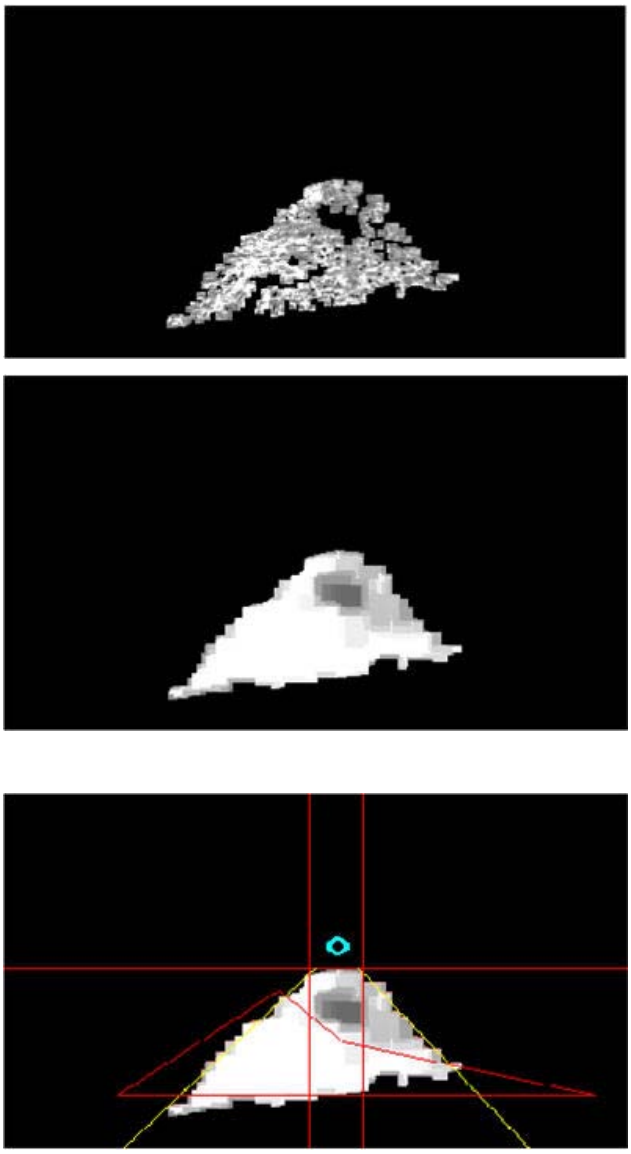


Figure 9: Starting with the clipped probability map (one), the image is first blurred (two), and the edges are fit to straight lines (three). Combining these lines with the effective horizon, a model for the road is constructed (four). A confidence score is assigned based on the number of pixels correctly classified by the implied road structure (four).

The algorithm presented has been tested against terrain types of widely varying difficulty. All of the images shown here were from a 320x240 camera image, and they were capable of being generated at ten frames per second on a computer based on the 1.7 GHz Mobile Intel Pentium 4 processor – M that was selected for its low-power requirements in the robot. For roads that are well defined, flat, and have a relatively uniform surface texture, the method allows extrapolation of the entire road to the horizon.

DISCUSSION

We started out describing recent developments with the OpenCV library [1] that allow it to be automatically optimized by using OpenCV with the IPP libraries [2], and we described how automatic optimization works through the cvswitcher function. To recap, the switcher is given a table of function identifiers and pointers to those functions. During launch, the switcher looks for the appropriate optimized IPP code for the machine it is running on and swaps in the address of those functions if the correct IPP module is found. Otherwise it retains the embedded optimized C functions. Because the source code for the switcher is available in the file cvswitcher.cpp, one may also override this functionality with custom functionality–by pointing it to one's own functions or alternate functions, depending on the processor used.

We then focused on a small part of OpenCV's support for learning-based vision. The Haar-feature-based boosted classifier cascade, based on the work of Viola and others [4, 5, 6] that is directly implemented in OpenCV, was described. When you download OpenCV, this classifier comes with a file that is trained to recognize faces and may be run by building and running the facedetect.c code that comes in the samples directory, which will normally be placed in C:\Program Files\OpenCV\samples\c on Windows machines when you install OpenCV. The procedure for using your own data to train a classifier cascade to recognize any object was then described. Using this, one can create tree, car, cat, or person detectors.

Note that one may make the classifier more flexible by using instead of, or in addition to, learning features from raw images; image processing can be used to enhance or pre-select images of features such as gradient magnitudes to be learned. This can help reduce lighting sensitivity, for example.

We then shifted gears to describe a project aimed at detecting and segmenting an abstract object–a road. This involved using laser range data to identify a polygon of a nearby road to use as a seed for learning and modeling clusters of road-colored pixels. This learned model was then used to segment the road beyond the range of the laser range finders. This is essentially a local model of what pixels "look" like road. Global geometric constraints are then used to only accept road segmentations that in fact look like the shape of a road as in Figure 9 bottom. The segmented road pixels are then projected along with the laser data into a bird's eye view 2D path planner. Vision significantly extends the range of road detection out from the lasers and so allows the robot to travel at much higher speeds. Using the local and global constraints together, false positive road identification was completely eliminated.

In the above, we just touched on the learning-based vision possibilities of OpenCV. Many other techniques are supported. For example OpenCV contains methods for gathering and matching histograms of image features; learning the means and variances of pixels over a moving window of time; Kalman and Particle ("Condensation") filters for tracking, and data that can be clustered or tracked with the Meanshift or CAMShift [8] algorithms. In Release version 1.0 several more machine-learning algorithms for computer vision should become available such as convolutional neural networks, general back propagation, general decision trees, several methods of statistical boosting, and random forests. These new algorithms will enable powerful approaches to learning-based vision.

CONCLUSION

Computer vision, unlike for example factory machine vision, happens in unconstrained environments, potentially with changing cameras and changing lighting and camera views. Also, some "objects" such as roads, rivers, bushes, etc. are just difficult to describe. In these situations, engineering a model *a-priori* can be difficult.

With learning-based vision, one just "points" the algorithm at the data and useful models for detection, segmentation, and identification can often be formed. Learning can often easily fuse or incorporate other sensing modalities such as sound, vibration, or heat. Since cameras and sensors are becoming cheap and powerful and learning algorithms have a vast appetite for computational threads, Intel is very interested in enabling geometric and learning-based vision routines in its OpenCV library since such routines are vast consumers of computational power.

REFERENCES

[1] Open Source Computer Vision Library: http://www.intel.com/research/mrl/research/opencv

[2] Intel® Integrated Performance Primitives http://www.intel.com/software/products/perflib

[3] Stewart Taylor, "Intel® Integrated Performance Primitives," in *How to Optimize Software Applications Using Intel® IPP*" http://www.intel.com/intelpress/sum_ipp.htm

[4] Paul Viola and Michael J. Jones, "Rapid Object Detection using a Boosted Cascade of Simple Features," *IEEE CVPR*, 2001.

[5] Rainer Lienhart and Jochen Maydt, "An Extended Set of Haar-like Features for Rapid Object Detection," Submitted to *ICIP2002*.

[6] Alexander Kuranov, Rainer Lienhart, and Vadim Pisarevsky, "An Empirical Analysis of Boosting Algorithms for Rapid Objects With an Extended Set of Haar-like Features," *Intel Technical Report* MRL-TR-July02-01, 2002.

[7] Freund, Y. and Schapire, R. E. (1996b), "Experiments with a new boosting algorithm," in *Machine Learning: Proceedings of the Thirteenth International Conference*, Morgan Kauman, San Francisco, pp. 148-156, 1996.

[8] Bradski, G., "Computer Vision Face Tracking For Use in a Perceptual User Interface," *Intel Technology Journal*,http://developer.intel.com/technology/itj/q21998/articles/art_2.htm, Q2 1998.

AUTHORS' BIOGRAPHIES

Gary Rost Bradski is a principal engineer and manager of the Machine Learning group for Intel Research. His current interests are learning-based vision and sensor fusion in world models. Gary received a B.S. degree from U.C. Berkeley in May, 1981. He received his Ph.D. degree in Cognitive and Neural Systems (mathematical modeling of biological perception) in May, 1994 from Boston University Center for Adaptive Systems. He started and was the technical content director of OpenCV working closely with Vadim. Currently, he consults on OpenCV and machine learning content with the performance primitives group. His e-mail is Garybradski at gmail.com.

Adrian Kaehler is a senior software engineer working in the Enterprise Platforms Group. His interests include machine learning, statistical modeling, and computer vision. Adrian received a B.A. degree in Physics from the University of California at Santa Cruz in 1992 and his Ph.D. degree in Theoretical Physics from Columbia University in 1998. Currently, Adrian is involved with a variety of vision-related projects in and outside of Intel. His e-mail is Adrian.l.Kaehler at intel.com.

Vadim Pisarevsky is a software engineer in the Computational Software Lab in Intel. His interests are in image processing, computer vision, machine learning, algorithm optimization, and programming languages. Vadim received a Masters degree in Mathematics from the Nizhny Novgorod State University, in 1998. He has been involved in different software projects related to multimedia processing since 1996. In 2000, he joined Intel Russia Research Center where he led the OpenCV development team for over four years. Currently, he is working in the software department and continues to improve OpenCV and its integration with Intel Integrated Performance Primitives. His e-mail is Vadim.Pisarevsky at intel.com.

Performance Scalability of Data-Mining Workloads in Bioinformatics

Yurong Chen, Corporate Technology Group, Intel Corporation
Qian Diao, Corporate Technology Group, Intel Corporation
Carole Dulong, Corporate Technology Group, Intel Corporation
Chunrong Lai, Corporate Technology Group, Intel Corporation
Wei Hu, Corporate Technology Group, Intel Corporation
Eric Li, Corporate Technology Group, Intel Corporation
Wenlong Li, Corporate Technology Group, Intel Corporation
Tao Wang , Corporate Technology Group, Intel Corporation
Yimin Zhang, Corporate Technology Group, Intel Corporation

Index words: data mining, bioinformatics, performance scalability analysis

ABSTRACT

Data mining is the extraction of hidden predictive information from large data bases. Emerging data-mining applications are important factors to drive the architecture of future microprocessors. This paper analyzes the performance scalability on parallel architectures of such applications to understand how to best architect the next generation of microprocessors that will have many CPU cores on chip.

Bioinformatics is one of the most active research areas in computer science, and it relies heavily on many types of data-mining techniques. In this paper, we report on the performance scalability analysis of six bioinformatics applications on a 16-way SMP based on Intel® Xeon™ microprocessor system. These applications are very compute intensive, and they manipulate very large data sets; many of them are freely accessible. Bioinformatics is a good proxy for workload analysis of general data-mining applications. Our experiments show that these applications exhibit good parallel behaviors after some algorithm-level reformulations, or careful parallelism selection. Most of them scale well with increased numbers of processors, with a speed-up of up to 14.4X on 16 processors.

® Intel and Xeon are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States and other countries.

We start with an introduction to data mining. The data-mining techniques studied are briefly described, and the selected workloads using these techniques are listed. We then provide a brief description of the methodology used for the studies. We present the scalability analysis of three workloads related to Bayesian Network (BN) structure, two workloads relevant to recognition, and one workload related to optimization. We conclude with the key lessons of the study. These workloads are compute intensive and data parallel. They manipulate large amounts of data that stress the cache hierarchy. Techniques optimizing the use of caches are key to ensure performance scalability of these workloads on parallel architectures.

DATA MINING: A DEFINITION

Databases today can hold terabytes of data that hide a lot of information. *Data mining* is the technology that draws meaningful conclusions, extracts knowledge, and acquires models from these data.

The potential returns of data mining are large. Innovative organizations worldwide use it to locate and appeal to higher-value customers, to reconfigure their product offerings to increase sales, and to minimize losses due to error or fraud. Data mining has been widely used in various domains such as retail, telecommunication, medical diagnosis, and financial services.

Bioinformatics Application Classification

Broadly speaking, bioinformatics is the recording, annotation, storage, analysis, and search/retrieval of nucleic acid sequences (genes and RNAs), protein sequences, and structural information. Currently, bioinformatics mainly includes databases of sequences and structural information, as well as methods to access, search, analyze, visualize, and retrieve the information. Bioinformatics applications can be categorized as follows:

- *Sequencing*: gene sequence assembly
- *Sequence alignment and search*: pair-wise and multiple sequence alignment, database search.
- *Sequence analysis*: gene finding, Single Nucleotide Polymorphisms (SNP) pattern analysis, etc.
- *Structure analysis and structural genomics*: protein secondary/tertiary structure prediction, protein folding.
- *Comparative genomics*: whole genome alignment, phylogenetic tree reconstruction.
- *Functional genomics/proteomics and system biology*: function prediction of non-coding sequences, gene expression clustering, and gene regulatory networks.
- *Clinical field* (gene expression classification, etc.)

Bioinformatics relies heavily on many types of data-mining techniques. For the purposes of our study, we describe several categories of data-mining techniques, and corresponding workloads.

DATA-MINING TECHNIQUES STUDIED

Data mining uses a variety of data analysis tools to discover patterns and relationships in data that may be used to make valid predictions. It takes advantage of advances in the fields of Artificial Intelligence (AI) and statistics. Algorithms that are employed in many areas such as pattern recognition, machine learning, decision-making support, and statistical modeling can be used in data mining. Following, we briefly introduce some of the techniques and algorithms.

Bayesian Networks

A Bayesian Network (BN) is a probabilistic model that encodes probabilistic relationships between variables of interest. Over the last decade or so, BNs have been widely used in statistics, machine learning, pattern recognition, engineering, diagnostics, decision making, and so on.

Learning the structure of a BN from data is the most important task of BN applications [1]. The goal is to identify the statistic relationship between variables, and usually at the same time the conditional probability distribution of each variable can also be determined. BN structure learning has become an active research area in recent years [2, 3, 4].

The most popular approach to structure learning is to turn it into an optimization exercise. We first introduced a scoring function to evaluate the network with respect to the training data and to output a value that reflects how well the network scores, relative to the available data. We then search through possible network structures for the best scored network and take this as the network learned from the data. In general, the search problem is NP-hard [5]. Most algorithms use heuristic search methods, such as the Markov Chain Monte Carlo (MCMC) sampling [1, 6], K2 [1], simulated annealing [2], etc., of which the greedy hill-climbing algorithm is the most efficient and popular approach.

We have studied three applications using BNs: SNPs [7], GeneNet [8], and SEMPHY [10]. All are a variation on the hill-climbing concept. In SNPs and GeneNet applications, all the training data are observed (i.e., there are no missing data), and a standard hill-climbing search algorithm is employed. In a SEMPHY application, only a part of the data is observed (i.e., there are some missing data), and the hill-climbing learning procedure is used with an EM parameter-learning procedure. The total algorithm is called a Structural EM Algorithm [11, 12]. We look at these three applications next.

Figure 1(a): An instance of the SNP sequences (the symbol * denotes the SNP site)

Workloads Studied Using Bayesian Networks

SNPs are DNA sequence variations that occur when a single nucleotide (A, T, C, or G) is altered at certain loci in the genome sequence (shown in Figure 1(a)). These variations are major sources of individual diversity. Understanding the importance of the recently identified SNPs in human genes has become a goal of human genetics [13]. A common understanding of the cause of SNPs is nucleotide substitution. A number of studies have shown that the substitution process can be context

dependent, that is, neighboring base composition can influence the substitution bias at a particular site. Substitution patterns at polymorphic sites and bias patterns in nucleotides neighboring polymorphic sites are important for understanding molecular mechanisms of mutation and genome evolution [14, 15].

This research suggests the existence of context dependencies near SNP sites. However, by employing BN structure learning, not only the dependencies around the SNPs loci can be confirmed, but also the dependencies model and influence strength for each loci neighboring the SNPs, can be discovered. The task can be formulated as follows: each locus on the sequence segment is represented as a discrete random variable of BN, with integer value ranges from 0 to 3 (each corresponds to A, C, G, or T), so the possible relations among these loci can be represented by the BN structure.

DNA microarray experiments measure all the genes of an organism, providing a "genomic" viewpoint on gene expression. Most of the analysis tools currently used are based on *clustering* algorithms. These algorithms attempt to locate groups of genes that have similar expression patterns over a set of experiments. A more ambitious goal for analysis is revealing the structure of the transcriptional regulation process. Thus, BN provides a natural approach to model the regulatory relationship between genes.

By representing each gene as a variable of the BN, the gene expression data analysis problem can be formulated as a BN structure-learning problem. The GeneNet application uses the same serial hill-climbing algorithm as the SNPs problem, but its input set has different characteristics: for SNPs, the BN contains only tens of variables (<100), but a large number of training data (typically 50K – 500K), while in GeneNet, there are many variables (1K – 10K typically), but only hundreds of training cases.

Unlike the previous applications on BN structure learning, the SEMPHY application uses the Structural Expectation Maximization (SEM) algorithm. SEMPHY differs from traditional BN applications in two aspects: it searches from a bifurcating tree space rather than the DAG space; and it can find the optimal solution based on missing data.

Classification and Prediction

To classify an object is to put it into a pre-defined class or category, or to assign it a label. Prediction can be viewed as the construction and use of a model to classify an unlabeled sample. Classification and prediction have numerous applications including credit approval, medical diagnosis, performance prediction, and selective marketing.

For example, the Support Vector Machine (SVM) has been considered a state-of-the-art classification technique since the 1990s, and we have used it in disease gene finding, based on the Support Vector Machines Recursive Feature Elimination (SVM-RFE) method.

Workloads Studied Using Classification Techniques

SVM-RFE [17] is a feature selection method to refine the optimum feature set by using SVM in a wrapper approach (shown in Figure 1(b)). It selects or omits dimensions of the data, depending on a performance measurement of the SVM classifier. It is much more robust to data overfitting than other methods, including combinatorial search. (In SVM-RFE, the induction algorithm used is SVM.)

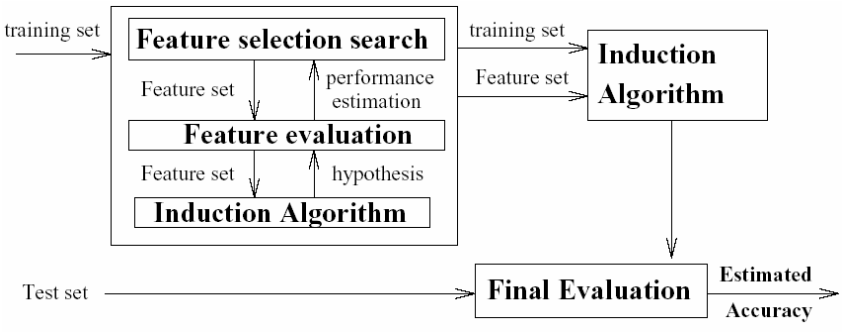


Figure 1(b): Overview of wrapper method for feature selection

In bioinformatics, SVM-RFE has been used for the task of microarray data analysis, particularly in disease gene finding. It eliminates gene redundancy automatically and yields better and more compact gene subsets. The selection is obtained by a recursive feature elimination process: at each RFE step, a gene is discarded from the active variables of an SVM classification model. The features are eliminated according to a criterion related to

their support for the discrimination function, and the SVM is re-trained at each step.

We now describe the second workload studied in this report using a classification technique: the Cocke-Younger-Kasami (CYK) algorithm. This technique uses a basic parsing algorithm for any context-free language, and it is used in RSEARCH during an RNA secondary structure homolog search. RSEARCH [13] uses Stochastic Context-Free Grammar (SCFG) to take a single RNA sequence with its secondary structure, and it utilizes the CYK algorithm to search a database for homologous RNAs through local alignment. RSEARCH has better performance in accuracy for RNA homolog search than other sequence search programs, such as BLAST and SSEARCH, and it is also capable of finding significant remote RNA structure homologies.

SCFG and its decoding algorithm CYK used in RSEARCH can also be applied in other areas, such as language modeling for speech recognition [14], language parsing for natural language processing [15], multitasked activities recognition for computer vision [16], and so on.

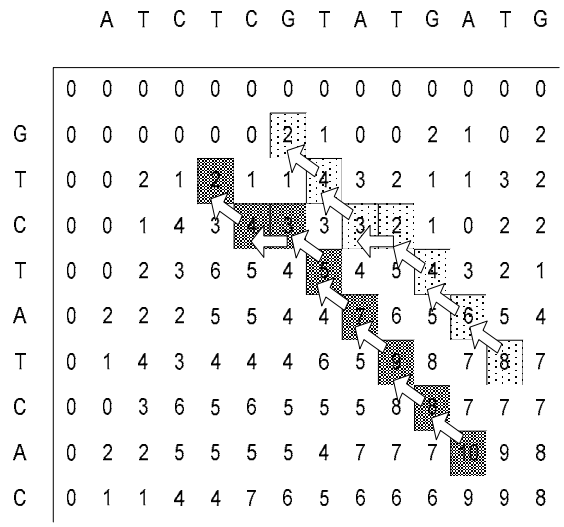
Optimization

Dynamic Programming (DP) is an approach developed to solve sequential, or multi-stage, decision problems. This approach is also applicable to decision problems where sequential property is induced solely for computational convenience. DP is widely used in combinatorial optimization, speech recognition, sequence alignment, time series data processing, etc. Even when it does not solve a problem completely, it can be useful as part of an overall approach. In particular, DP plays an important role in solving similarity problems of some major data-mining tasks, such as Association Rule Mining (ARM), similar time sequence mining, similar image mining, and so on. ARM is a matter of looking for association rules in data. An association rule is an expression X IMPLIES Y, where X and Y are sets of items. The intuitive meaning of such a rule is that transactions of the database that contain X tend to contain Y.

Workloads Studied Using Optimization Techniques

Sequence alignment is an important tool in bioinformatics, text, acoustic signal, and image processing. It is capable of identifying the similar and diverged regions between two sequences, e.g., biological DNA/protein sequences or text strings. From a biological point of view, matches may turn out to be similar functions, e.g., homology pairs and conserved regions, while mismatches may detect functional differences, e.g., SNP.

With DP, Needleman and Wunsch presented the first global alignment algorithm in 1970 [18]. Smith and Waterman improved this algorithm for the local alignment to find the longest common substring [19] (shown in Figure 2). In this paper, we study an efficient Parallel Linear Space Alignment (PLSA) for large-scale sequence alignment. By introducing the novel grid cache, global/local start points, the algorithm reduces the re-computations of the trace-back period dramatically, and it provides more parallelism than other methods do. Besides the algorithms mentioned above, there are many other techniques or algorithms that have been widely used in data mining, such as clustering, statistic modeling, association rules mining, Naïve Bayes classifiers, neural networks, memory-based reasoning, evolutionary programming, regression, decision trees, etc.



Similarity matrix of the alignment of sequence ATCTCGTATGATG and GTCTATCA. k=2, substitution cost of +2 if the two characters are identical and -1 otherwise. Both the first and extension gap penalty are -1. Two traceback paths deliver the non-intersecting optimal and near-optimal local alignments.

Figure 2: Smith Waterman sequence alignment

SUMMARY OF APPLICATIONS STUDIED

Table 1 summarizes the six applications studied in this paper. We list the type of algorithm they use, how the parallelism can be exploited, and we show what types of applications they are representative of.

We have used applications developed in universities. For example, we have used SEMPHY (from the Hebrew University in Jerusalem) for reconstruction of phylogenetic trees; and RSEARCH (from Washington University in St. Louis) for RNA secondary structure homolog search. We have also developed some workloads. For instance, we have used a BN structured learning to solve the discovering of patterns in SNPs, and to analyze gene expression data in DNA microarrays in GeneNet. In PLSA, we have developed a novel large-scale alignment algorithm for the whole genome alignment.

This section describes the methodology used to parallelize, optimize, and analyze the workloads. The first step is to profile the workloads to identify the hot spots. The Intel® VTune™ Performance Analyzer was used for function-level profiling, and for correlation of the hardware performance events with the source code.

Parallelization was done with the Open MP programming model that is well suited to exploit the data parallelism of these algorithms. For example, in RSEARCH, the whole RNA sequence database is scanned with a three-dimensional dynamic programming algorithm.

Generally, the query RNA sequence is far shorter than the sequences in the database. RSEARCH first defines a value "D_scale," representing the largest ratio between the match part of the sequence and the query sequence. The database sequence is segmented into different subsets that overlap so that the D_scale can be a multiplier of the query length. The search through the different subsets can be done in parallel by different processors.

The costs of synchronization, locks, and barriers were measured using the Intel VTune Performance Analyzer thread profiler for OMP applications. The experiments show that for most studied applications these costs are very low, which is expected for data-parallel applications with very little synchronization between threads.

After having characterized the communication overhead, synchronizations (explicit or implicit), and load-balancing performance (dynamic or static partitioning methods), we measured the performance increase on up to 16 processors, and we characterized the memory hierarchy behavior.

® Intel and VTune are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States and other countries.

Table 1: Bioinformatics workloads, algorithms, applicability Workload Analysis Methodology

Category	Workload	Algorithm	Parallelism	Applicability
Bayesian Network/Structure Learning	SNPs (Single Nucleotide Polymorphisms)	Structure learning Hill-climbing	Data parallelism (instance data, node, tree)	• Pattern recognition • Speech recognition • Optimization • Text mining • Game • Decision Making
	GeneNet (Gene Expression analysis in microarrays)			
	SEMPHY	Structural EM		
Classification and Prediction	RSEARCH (homologous RNA sequence)	Stochastic Context Free Grammar: CYK Local alignment	Data base Segmentation	• Recognition • Classification • Prediction • Speech recognition, language parsing
	SVM-RFE (Disease Gene Finding in Microarrays)	SVM based feature selection	Data blocking matrix/ vector multiply	• Pattern recognition • Classification • optimization
Optimization	PLSA (Parallel Linear Space Alignment)	Dynamic Programming	Data blocking Wave-front parallelism	• Pattern recognition, • Text mining • Association Rule Mining • Combinatorial optimization

To measure the performance of our workloads, we use a 16-way SMP based on Intel Xeon microprocessor system interconnected with a crossbar. The configuration of the 16-way system is described in Table 2. The machine is running the SUSE ES Linux* operating system environment for all the experiments. All applications were compiled with the Intel Compiler v8.0, at the highest level of optimization.

PERFORMANCE SCALABILITY ANALYSIS

All the workloads studied in this paper use data parallelism, where all processors are executing the same code on different data. Figure 3 shows most of the workloads scale well with increased numbers of processors, and two workloads exhibit linear speed-up performance (SEMPHY, PLSA).

* Other brands and names are the property of their respective owners.

Table 2: Configuration of the 16-way SMP based on Intel® Xeon™ microprocessor system

Processor Speed	3.0 GHz
L1 Data Cache	8 KB, hit latency: 2 cycles
L2 Unified Cache	512 KB, hit latency: ~10 cycles
L3 Unified Cache	4 MB, hit latency: 30+ cycles
L4 on-board Unified Cache	32 MB, hit latency: 300+ cycles
Interconnection	Crossbar
System Bus Speed	400 MHz
Front Side Bus Bandwidth	3.2 GB/s
Memory Size	8 GB, dual-channel DDR 400

To understand the performance-limiting factors, we have quantified the parallelism overhead such as synchronizations penalties, load imbalance, and sequential sections. They are not significant, especially for large data sets typical in current, and future, data-mining workloads. Figure 4 shows these metrics for two selected workloads, where the sequential area and the load imbalance penalties are diminishing when the data set size increases. These results are typical of all the workloads studied in this report. Very few synchronizations are needed between threads, and load balancing between threads is not an issue. In SEMPHY, for example, computations are distributed in four kernels that are nested loops, with no dependency between loop iterations. The basic "Parallel For" pragma is used to parallelize these loops. In one of the kernels, the data decomposition is constrained by the relation between the number of leaves in the tree, and the length of the DNA or protein sequence. This creates a small load imbalance for small data sets. The balancing issue goes away for large data sets, where the constraints become insignificant.

To identify the performance bottlenecks, we characterize the memory hierarchy behavior by measuring the cache miss rates and the Front Side Bus (FSB) bandwidth. In Figure 5, it is interesting to see that the L2 cache miss rates vary very little with the number of processors. The data sets are large enough not to fit in L2, even when problems are divided among 16 processors. We can observe SVM-RFE has very high L2 and L3 cache miss rates. The function profile of SVM-RFE shows that the SVM training is the most time-consuming kernel. It consists of a large number of vector-vector multiplications. We have used the Intel Math Kernel Library (MKL) for these operations to take advantage of its highly optimized routines. But there is no data reuse in vector-vector multiply operations, and this explains the high cache miss rates. With increasing numbers of processors, the shared system bus sees high memory traffic, resulting in high memory latencies. This limits the speed-up performance for the SVM-RFE workload. In the case of SNP, and GeneNet, the L3 miss rates are modest, but they increase with the number of processors, when there are more than four processors. These applications have at least one large data structure shared between the thread. This data structure is shared efficiently between four processors in the L4 cache of the 16-way system. But sharing is done through the main memory interconnect for more than four processors, and this limits the scalability of these workloads.

Figure 6 shows the cache miss per instruction in L2 and L3 and the FSB bandwidth used by these workloads running a single thread.

We have verified that these miss per instructions in L2 and L3 remain about constant per workload as the number of threads increases from 1 to 16.

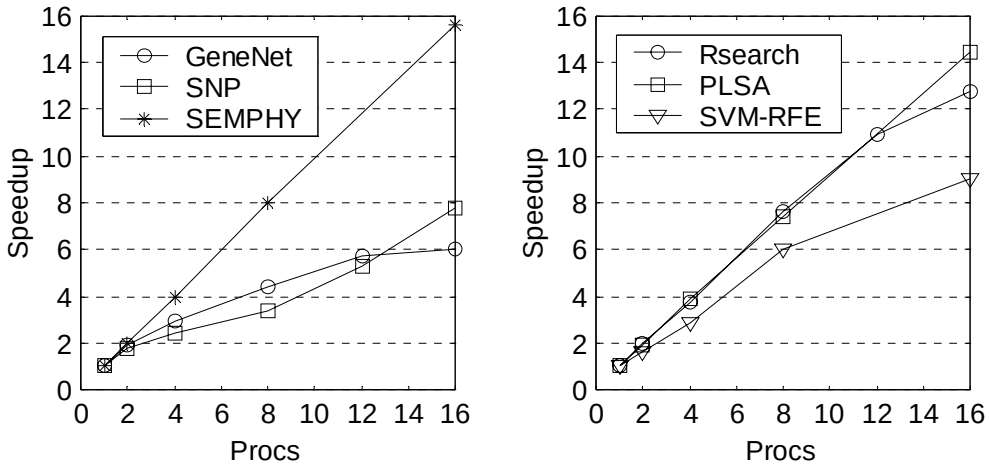


Figure 3: Performance speed-up as a function of the number of processors

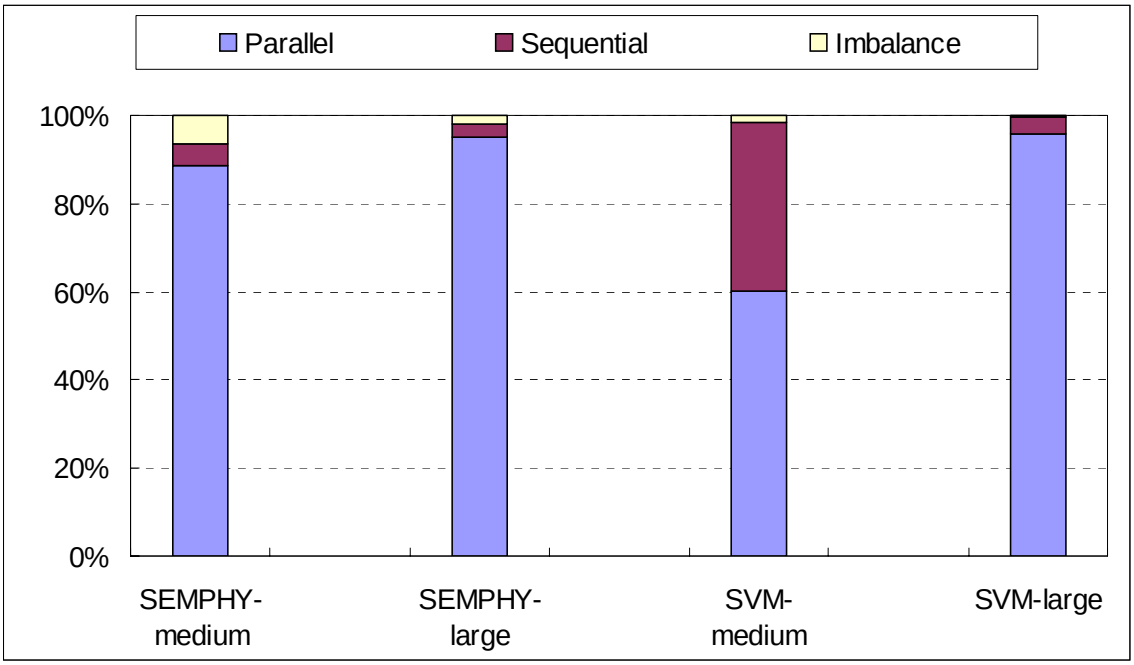


Figure 4: Distribution of time spent in parallel, and sequential, code

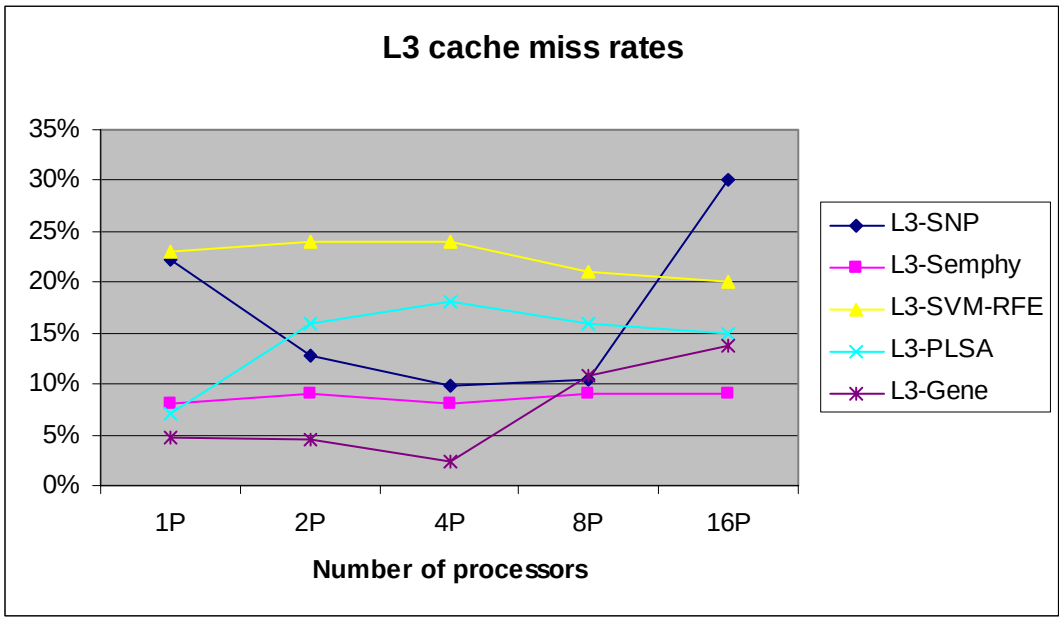


Figure 5: L2/L3 cache miss rates as a function of the number of processors

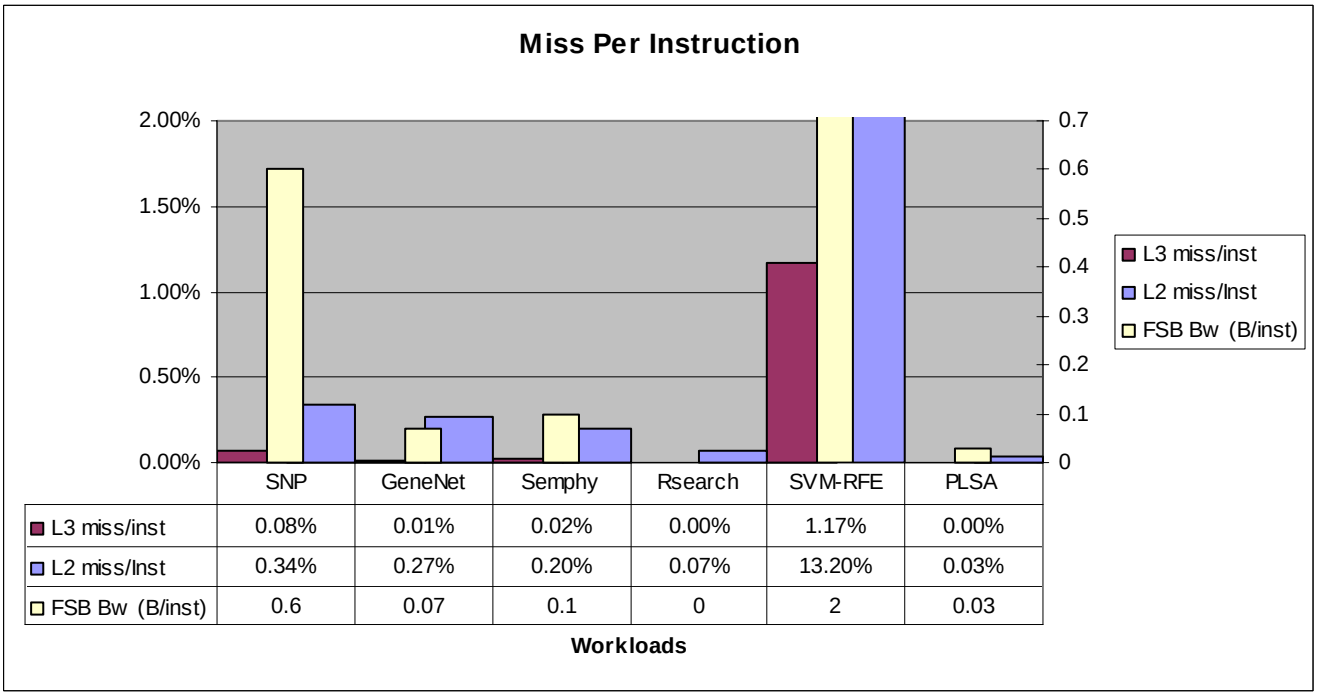


Figure 6: L2 and L3 miss per instruction for the one processor case

Most of these workloads are integer workloads, and they use very few floating-point operations. So the bandwidth per instruction shown in Figure 6 is in byte-per-integer operation. The three workloads that have the lowest L3 miss per instruction are the ones whose performance scales almost linearly with the number of processors: SEMPHY, RSEARCH, and PLSA. The high miss per instruction in both L2, and L3 in SVM-RFE, has already been explained by the vector-vector multiply operations used most of the time in SVM-RFE, and the fact that there is no data reuse. Note that the Intel Xeon processor prefetchers are highly effective for such operations working on contiguous data, and they explain why the L3 miss per instruction is not higher than it is. For SNP and

GeneNet, the cache misses per instruction are higher than for the workloads scaling very well, but much lower than for SVM-RFE. Yet these two workloads do not scale as well as SVM-RFE. Caches are not the limiting factors for these workloads. Threads spend a significant part of the time waiting for each other at the end of parallel sections, because of load balancing. In SNP for example, parallel threads perform the hill-climbing algorithm on different input data. The number of computations depends on the data structure, and the difference in computation requirements between threads explains the load imbalance.

Bus bandwidth utilization varies widely between these six workloads. They go all the way from virtually 0 for RSEARCH that is highly compute intensive, with very high reuse of data in cache, to 2 Bytes per instruction for SVM-RFE whose main computation is a vector-vector multiply that has very little data reuse.

Figure 7 shows how the bus bandwidth utilization varies with the number of processors. It uses a log scale on both axis. The bandwidth used by the different workloads varies widely between workloads, which is explained by the very different miss rates the workloads get in the L2 and L3 caches. But for all workloads, the FSB bandwidth varies linearly with the number of processors. This indicates that the bus bandwidth is not a limiting factor on this system.

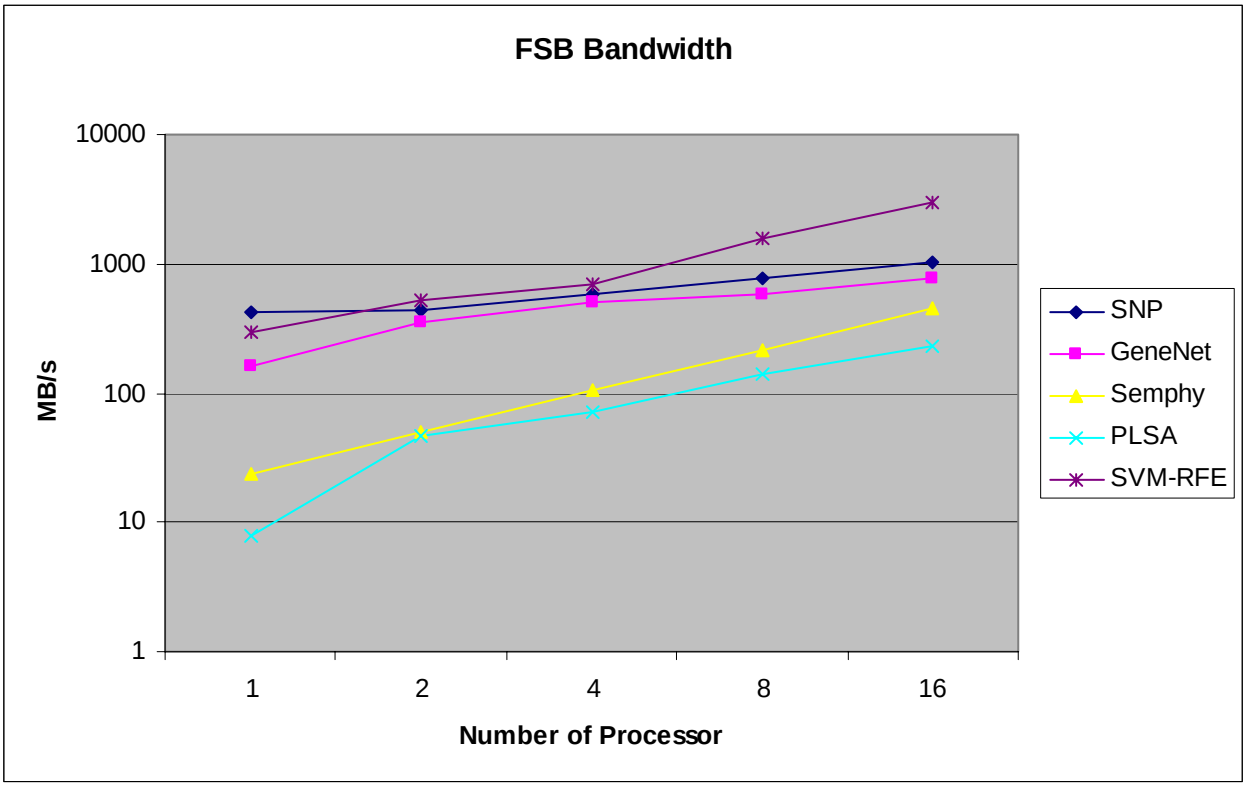


Figure 7: FSB bandwidth as a function of the number of processors

CONCLUSION

The bioinformatics workloads studied in this paper are representative of general-purpose data-mining techniques. They use BNs, SVMs, and DP, among other methods. They are data parallel workloads with large data sets and are very compute intensive.

Performance scalability up to 16 processors is very good for some workloads such as SEMPHY, RSEARCH, and PLSA that exhibit almost linear speed-up. It is not as good, but quite respectable, for SNP, GeneNet, and SVM-RFE.

The FSB bandwidth is not a limiting factor for performance scalability (for the system used in the study). The FSB utilization grows linearly with the number of active processors for all workloads. Caches are too small, which leads to the high miss rates of 5% to 30% in the L3 cache. The large latencies to the L4 cache and to main memory DRAM are limiting scaling for workloads like SVM-RFE and SNP that access large and complex data structures.

ACKNOWLEDGMENTS

We acknowledge the encouragement and help that we have received from Bob Liang. Additional thanks go out to individuals who have reviewed the paper and provided valuable feedback: Chu-cheow Lim, Gideon Gerzon, and Chuck Yount.

REFERENCES

[1] Kevin Murphy, "The Bayes Net Toolbox for Matlab," Computing Science and Statistics, vol. 33, 2001.

[2] D. Heckerman et al., "Learning Bayesian networks: the combination of knowledge and statistical data," Technical Report MSR-TR-09-09, Microsoft Research, 1994.

[3] D. Heckerman, "A Tutorial on Learning with Bayesian Networks," in Learning in Graphical Models, M. Jordan, ed. MIT Press, Cambridge, MA, 1999.

[4] W. Buntine, "A guide to the literature on learning probabilistic networks from data," IEEE Trans. On Knowledge and Data Engineering, 8:195-210, 1996.

[5] Chickering, D. M., "Learning Bayesian networks is NP-Complete," in D. Fisher and H. Lenz (Eds.), Learning from data: Artificial intelligence and statistics v, pp. 121–130.

[6] P. Guidici et al., "Markov Chain Monte Carlo methods for probabilistic network model determination," Journal of the Italian Statistical Society, 7, pp. 171-183.

[7] X. Ma et al., "Discovering Possible Context Dependencies around SNP Sites in Human Genes with Bayesian Network Learning," Eighth International Conference on Control, Automation, Robotics and Vision (ICARCV), 2004.

[8] N. Friedman et al., "Using Bayesian Networks to Analyze Expression Data," Journal of Computational Biology, 7:601--620, 2000.

[9] Bateman, A. et al., "The Pfam Protein Families Database," Nucleic Acids Research Database, Issue 32:D138-D141. 2004.

[10] N. Friedman, "A Structural EM algorithm for Phylogenetic Inference," Journal of Computational Biology, 9:331-353, 2002.

[11] N. Friedman, "Learning belief networks in the presence of missing values and hidden variables," Fisher, D. ed., Proceedings of the Fourteenth International Conference on Machine Learning, pp. 125-133, Morgan Kaufman, San Francisco, (1997).

[12] "The Bayesian structure EM algorithm," in Fourteenth Conf. on Uncertainty in Artificial Intelligence (UAI), 1998.

[13] Klein, R.J. and Eddy, S.R., "RSEARCH: Finding homologs of single structured RNA sequences," BMC Bioinformatics, 2003, 4:44.

[14] Jurafsky, D., Wooters, C., Segal, J., Stolcke, A., Fosler, E., Tajchman, G., Morgan, N., "Using a Stochastic Context-Free Grammar as a Language Model for Speech Recognition," in Proc. ICASSP'95, 189-192.

[15] Fujisaki, T.; Jelinek, F.; Cocke, J.; Black, E.; and Nishino, T., "A probabilistic parsing method for sentence disambiguation," in Current Issues in Parsing Technology, edited by Masaru Tomita, Kluwer Academic Publishers, 1991, pp. 139–152.

[16] Darnell Moore, Irfan Essa, "Recognizing Multitasked Activities using Stochastic Context-Free Grammar," in Proceedings of Workshop on Models versus Exemplars in Computer Vision," held in Conjunction with IEEE CVPR 2001, Kauai, Hawaii, December 2001.

[17] E. Osuna, R. Freund, and F. Girosi, "Training support vector machines: An application to face detection," in Proceedings of CVPR'97, pages 130–136, New York, NY, 1997b.

[18] Saul B. Needleman and Christian D. Wunsch, "A General Method Applicable to the Search for Similarities in the amino acid Sequence of Two Sequences," Journal of Molecular Biology, 48:443–453, 1970.

[19] Temple F. Smith and Michael S. Waterman, "Identification of Common Molecular Subsequences," Journal of Molecular Biology, 147:195–197, 1981.

AUTHORS' BIOGRAPHIES

Yurong Chen is a researcher at Microprocessor Technology Lab, Beijing. Currently he conducts research on emerging computing paradigms, parallel algorithms, and scalable workload development and analysis. He joined Intel in 2004. Before that he spent two years doing postdoctoral research work in computer science in the Institute of Software, Chinese Academy of Sciences. He received his BS degree in Applied Mathematics in 1998 and MS and PhD degrees in computational mathematics in 2002, all from Tsinghua University, China. His e-mail is yurong.chen at intel.com.

Qian Diao is a researcher in the Microprocessor Technology Lab, Beijing. Currently she works on various

data-mining techniques. In Intel, she has been involved in several projects related to information retrieval, speech recognition, visual tracking, bioinformatics, and statistical computing. She got her PhD. from Shanghai Jiao Tong University in 2000 and joined Intel in 2000. Her e-mail is qian.diao at intel.com.

Carole Dulong is a senior researcher and Computer Architect in the Microprocessor Technology Lab. She leads a team of researchers working on various data-mining techniques. She joined Intel in 1990. She was a member of the IPF architecture definition team and contributed to the IPF compiler design. She graduated from Institut Superieur d'Electronique de Paris (France). Her e-mail is carole.dulong at intel.com.

Wei Hu is a researcher in the Microprocessor Technology Lab, Beijing. Currently he works on various data-mining techniques. At Intel, he has been involved in several projects related to natural language processing and speech recognition, and statistic computing. He got his Ph.D from the Institute of Computing Technology, China Academy of Sciences (CAS/ICT) in 1998, and he joined Intel in 2000. His e-mail is wei.hu at intel.com.

Chunrong Lai is a researcher in the Microprocessor Technology Lab, Beijing. He is currently working on data-mining workload scalability and related architecture research within the Scalable Statistical Computing Group. His projects in Intel include various applications analysis, parallelization and optimization, performance simulation, and compiler re-target. He received his M.S. degree from the Chinese Academy of Sciences in 2000 and joined Intel as his first job. His e-mail is chunrong.lai at intel.com.

Eric Li is a researcher in the Microprocessor Technology Lab, Beijing. Currently he is working on algorithmic and workload analysis on data-mining applications. Prior to this position, he worked on the workload optimization, parallelization, analysis, and related algorithm methodology development. He received his M.S. degree from Tsinghua University in 2002, and joined Intel that same year. His e-mail is eric.q.li at intel.com.

Wenlong Li is a researcher in the Microprocessor Technology Lab, Beijing. Currently he is working on algorithmic and workload analysis on data-mining applications. Before this, he did research in loop compilation techniques for IPF architecture. He received his Ph.D degree from Tsinghua University in 2005 and joined Intel that same year. His e-mail is wenlong.li at intel.com.

Tao Wang is a researcher in the Microprocessor Technology Lab, Beijing. Currently he conducts research on data mining, machine learning, and computer vision techniques. At Intel, he has been involved in several projects related to visual tracking, bioinformatics, and content-based image/video retrieval. He received his Ph.D. degree from Tsinghua University in 2003 and joined Intel that same year. His e-mail is tao.wang at intel.com.

Yimin Zhang is a researcher in the Microprocessor Technology Lab, Beijing. He leads a team of researchers working on various statistical computing techniques and their scalability analysis. He joined Intel in 2000. At Intel, he has been involved in several projects related to natural language processing and speech recognition, especially focusing on Chinese-named entity extraction and DBN-based speech recognition. He received his B.A. degree from Fudan University in 1993, his M.S. degree from Shanghai Maritime University in 1996, and his Ph.D. degree from Shanghai Jiao Tong University in 1999, all in Computer Science. His e-mail is yimin.zhang at intel.com.

Performance and Scalability Analysis of Tree-Based Models in Large-Scale Data-Mining Problems

Alexander Borisov, Technology and Manufacturing Group, Intel Corporation
Igor Chikalov, Technology and Manufacturing Group, Intel Corporation
Victor Eruhimov, Corporate Technology Group, Intel Corporation
Eugene Tuv, Technology and Manufacturing Group, Intel Corporation

Index words**:** machine learning, data mining, decision trees

ABSTRACT

Statistical information processing is needed for many applications to extract patterns and unknown interdependencies between factors. A wide variety of data mining algorithms has been developed over the last decade, but active human intervention is still required to drive an analysis. The intention of expert work is to sequentially clarify models, and to compare models to provide accurate predictions. The productivity of expert work is largely constrained by the amount of time that is needed to compute model updates.

Recent modeling techniques such as classification and regression trees, and ensembles of machine-learning classifiers, incur high computational loads. Building such models in online interactive mode is a challenging task for upcoming platforms.

Tree-based models are applicable to a wide range of problems that include medical expert systems, analysis of manufacturing data, financial analysis, and market prediction. Ensembles of trees are notable for their accuracy. They can handle mixed-type data (consisting of both numerical and categorical data) and missing values. Several commercial packages implement these techniques.

In this paper we consider several data-mining methods based on ensembles of trees. The balance between complexity and accuracy is studied for different parameter sets. We provide an analysis of the computational resources required by the algorithms, and we discuss how they scale for execution on multiprocessor systems with shared memory.

INTRODUCTION

Fast growth and development of digital devices have resulted in a constantly increasing volume of digital data. According to the "How Much Information? 2003" survey [1], the world's total production of information content during 2002 required about 5 million Terabytes to store. More than 90% of this information is stored in electronic form, mostly on hard drives. This enormous amount of information creates a demand for fast and intelligent solutions for data-processing tasks. Many of these tasks can be approached using machine-learning techniques. Machine learning focuses on detecting and recognizing complex patterns in data. Examples are found in biometrics (fingerprint recognition, face recognition, machine vision), network security (intrusion detection), manufacturing (excursion analysis, statistical process control), financial analysis (trend prediction), medical systems (MRI scan analysis [2]), and forensic applications (genetic data analysis). Although these problems belong to different domains, the algorithms solving them have much in common. One of the core concepts commonly used for learning multidimensional patterns from data is a decision tree.

It is difficult to overestimate the influence of decision trees in general and Classification and Regression Trees (CART) [3] in particular on machine and statistical learning. CART has practically all the properties of a universal learner: it is fast, supports both discrete and continuous variables, elegantly handles missing data, and is invariant to monotone transformations of the input variables (and therefore resistant to outliers in input space). Another key advantage of CART is its embedded ability to select important variables during tree construction.

The main limitation of CART is relatively low prediction power. Intensive development of model averaging methods [4-8] over the last decade resulted in a series of very accurate tree-based ensemble techniques. A *tree ensemble* can be understood as a committee, where each member has a vote and the final decision is made based on the majority vote. The two most recent advances in tree ensemble techniques, gradient boosting tree (GBT) [9, 10] and Random Forest (RF) [11], have been proven to be among

the most accurate and versatile state-of-the-art learning machines. GBT serially builds tree ensembles where every new expert construction relies on previously built trees. RF builds trees independent of each other on a randomly selected subset of the training data, and predicts by majority vote (or average in regression).

In the next section we explain the details of the decision tree construction, and various learning algorithms associated with it. We then discuss applications that use decision trees, and we analyze their performance and scalability.

LEARNING WITH DECISION TREES

In supervised machine learning, we are given a dataset with a set of variables or attributes, often called "inputs" or "predictors," and a corresponding target, often called "response" or "output" values. The goal is to build a good model or predictive function that predicts unknown, future target values for given input values. When the response is numeric, the learning problem is called "regression." When the response takes on a discrete set of non-ordered categorical values, the learning problem is called "classification."

Single Tree

Decision trees are one of the most popular universal methods in machine learning/data mining and are commonly used for data exploration and hypothesis generation. CART is a commonly used decision tree algorithm [3]. It uses greedy, top-down recursive partitioning to divide the domain of input variables into sets of rectangular regions. These regions are as homogeneous as possible with respect to the response variable, and they fit a simple model in each region, either by majority vote for classification, or as a constant value for regression. At every step, a decision tree uses exhaustive search, by trying all combinations of variables, and split points to achieve the maximum reduction in impurity. Each split selection requires $O(kn\log n)$ operations, where k is the number of variables and n is the number of training samples.

A single decision tree can be visualized, interpreted, and tuned by an expert. The main limitations of CART are low accuracy and a high contribution to prediction error variance. High variances result from the use of piecewise, constant approximations.

Case Study 1

Figure 1 shows an example of a decision tree constructed to fit a functional dependency depicted in Figure 2a (regression problem with one response and two numeric predictors). The resulting piecewise approximation is shown in Figure 2b.

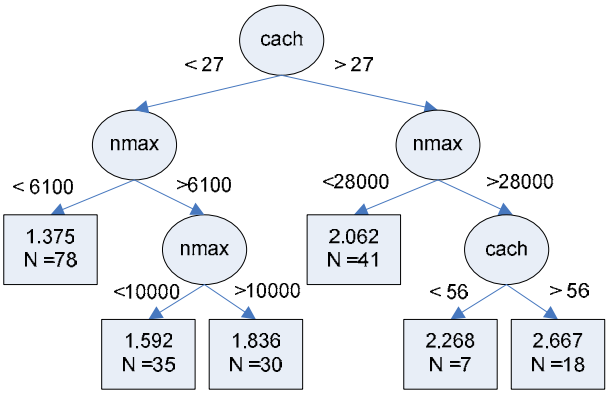


Figure 1: Example of a single regression decision tree

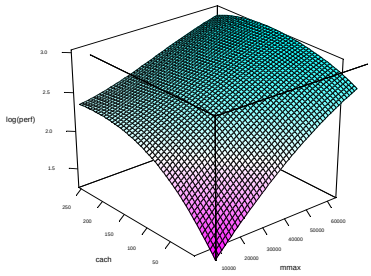


Figure 2(a): Example of a regression tree, original

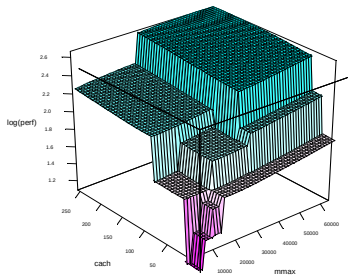


Figure 2(b): Example of a regression tree, CART prediction

Case Study 2

Figure 3 shows a decision tree learned from a car dataset where the response is a country where a car was produced, and predictors are reliability, horse power, mileage, and price. The split (characterized by a variable and its split value) at each node is chosen in order to maximize the number of samples (cars) that correspond to one of the countries. Note that the tree is built so that the presence of a single country in each node (indicated both by the width of the color bar and the text inside the node) grows from

the top of the tree to the bottom. Figure 4 illustrates the properties of the top node of the tree from Figure 3 and the properties of the corresponding split. The top part of the figure shows the table that compares the best split (the first row) to others. For each variable two numbers are reported (left to right): how a split on this variable reduces data impurity in comparison with the best split and how it is similar to the best split in data separation. Similar ("surrogate") splits are used for treating missing values–whenever the value of the variable corresponding to the primary split is missing, the surrogate split is used. The bottom part of Figure 4 shows the box plots of the variable values corresponding to the primary split (Reliability) for each of the response (Country) values. The red horizontal line corresponds to the split value. Note that it cuts off the high values of reliability corresponding to several countries ("Japan" and "Japan/USA").

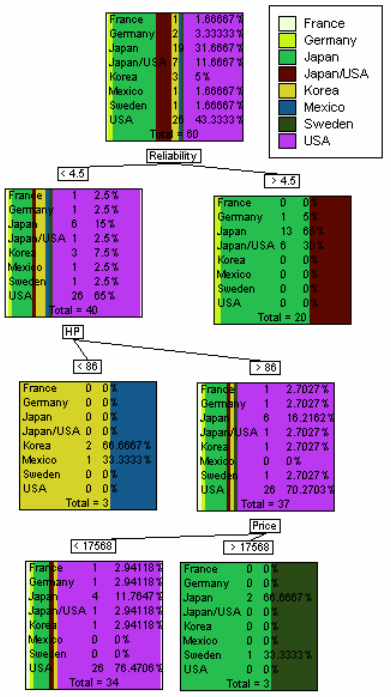


Figure 3: Example of a classification tree

Ensembles of Trees

Ensembles of trees combine outputs from multiple trees and can dramatically improve the performance of the resulting committee. There are two primary approaches to ensemble construction: parallel and serial. A *parallel ensemble* combines independently constructed trees, and therefore targets variance reduction.

In *serial ensembles*, every new constructed tree relies on previously built trees so that the resulting weighted combination of them forms an accurate learning engine. A serial ensemble algorithm is often more complex, but it is targeted to reduce both bias and variance, and usually shows excellent performance.

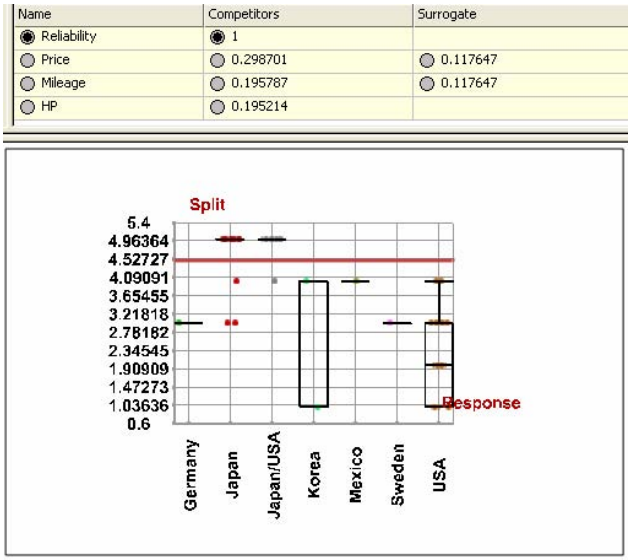


Figure 4: Split competitors and surrogate weights (upper part) and boxplots of split variable (Reliability) vs. response (Country)

APPLICATIONS

Decision tree-based learning is used in a wide range of applications. This list includes experimental data analysis in medicine and physics [3], market and customer analysis [12], manufacturing data exploration [13], and automated spam detection [14]. Single decision trees lack precision in predicting the data but are easier to interpret. For example, Figure 3 shows a tree for a cars dataset. It explains the connection between predictor and response variables using simple and interpretable rules. On the other hand, ensembles of decision trees have much better prediction accuracy, but they lack interpretability. Still, the model can be interpreted with techniques as described in [13].

There are several usage models for tree-based ensembles. For spam detection, the model is learned once and then used to predict the response in real-time with infrequent re-learning. For interactive data analysis, the model is built and re-learned interactively so that an analyst can experiment with parameters, and see the impact of the changes.

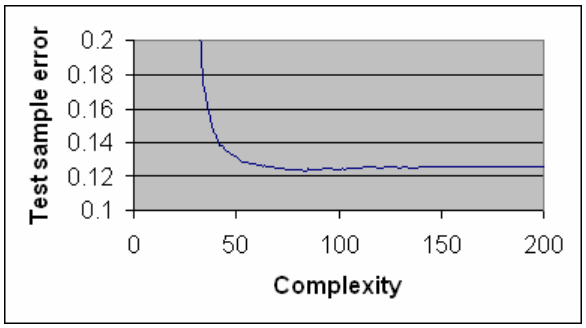


Figure 5: Serial ensemble complexity vs. error

Learning a decision tree ensemble from a large dataset is a computational challenge. For example, GBT ensemble learning on a manufacturing dataset that contains 200,000 samples, 129 predictors (mostly numeric), and binary response takes about eight minutes on a machine with a 3.06 GHz Intel® Xeon™ processor. The resulting ensemble consists of about 70 trees. Figure 5 shows the prediction error of the GBT ensemble depending on its complexity, measured as the number of trees. The optimal size of the ensemble corresponds to the minimum of the prediction error. The time to learn the serial ensemble is roughly proportional to the model complexity. Figure 6 shows the dependence of the learning time on the number of samples in the dataset. Both training data size and model complexity can increase the prediction power of the ensemble but the learning time will also increase.

In the next section, we investigate the computational properties of the parallel ensemble learning algorithm and explain how it can take advantage of the Symmetric Multiprocessor (SMP)-like architecture.

WORKLOAD ANALYSIS

In this section, we describe the requirements for resources used by the algorithm to construct an ensemble of classification trees. We give a description of data structures and methods used to optimize computations. Firstly, *hot operations* are identified that incur the main computational load. Secondly, an interaction with the Arithmetic Logic Unit (ALU), the memory subsystem, and the instruction decoder are described on an example data set. Finally, we present a scheme for parallelizing the algorithm for an SMP system as a function of the number of threads.

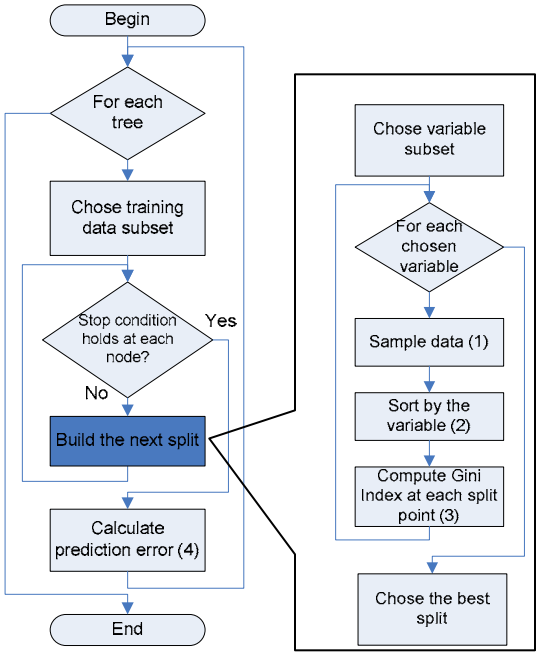


Figure 6: Block scheme of the Random Forest algorithm

Computing Platform Model

A complexity analysis of an algorithm requires a computational model. We consider a simplified model of a modern computer that consists of the following components:

- An ALU that executes logical, integer, and floating-point operations.
- A memory subsystem that consists of the main memory, several caches, and a system bus.
- An instruction decoder that includes a branch predictor.

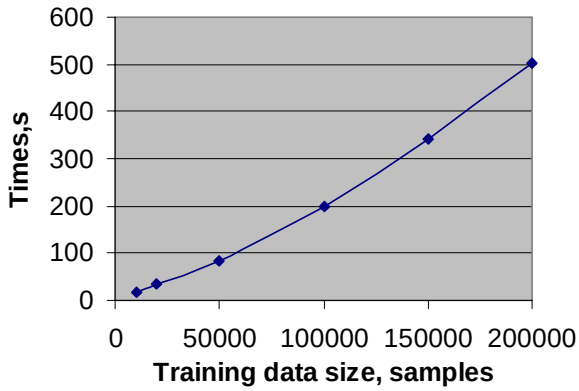


Figure 7: Learning time for a serial ensemble vs. the training data size

® Intel and Xeon are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States and other countries.

In the analysis we investigate the following problems:

- Where is the bottleneck, i.e., which component slows down the overall system performance?
- How many clock ticks are required to execute one instruction?
- What is the traffic to the main memory and each level of cache?
- Does the decoder manage to translate instructions in time, and how do unpredicted branches impair the performance.
- What is the expected performance improvement when the algorithm is executed on an SMP system.

Experimental Results

The training data set consists of 80,000 samples in the feature space that contains 127 numeric and 5 categorical variables, unless stated otherwise. One of the categorical variables is modeled by the Parallel Ensemble Learning (PEL) algorithm.

The following characteristics were measured on a computer with four 1.9 GHz Intel Xeon processors with hyper-threading technology and 3.5 Gb of RAM. The data are presented for a single thread (serial version), unless stated otherwise.

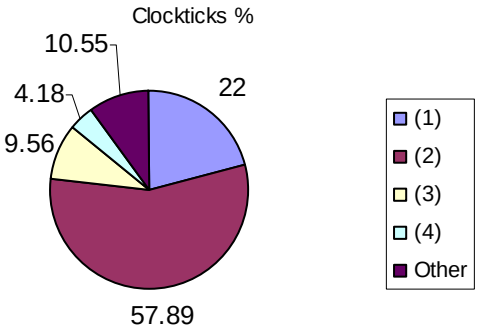


Figure 8: Distribution of execution time by operations

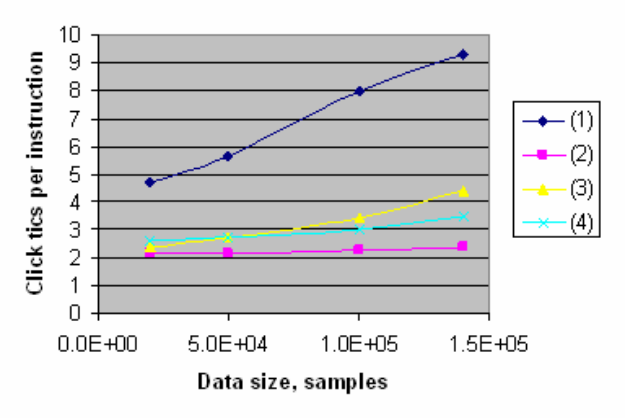


Figure 9: Clock ticks per retired instruction for functions (1)-(4) depending on the number of samples in the training dataset

Hot operations were identified in the optimized code. The main computational complexity is due to the following operations (also see Figure 6):

(1) Sampling a subset of data for calculating the current split. We randomly select both data samples and variables subset (the number of variables to be used in split calculation is a square root of the total number of variables in the dataset).

(2) Sorting training samples on each variable selected for the split. The sorting takes place for each variable independently and thus operates with a relatively small amount of data.

(3) Iterative calculation of data impurity reduction in order to find the best split variable and value. The calculation of the optimal split value is done using exhaustive search that is performed with one pass through the sorted array of values.

(4) Propagate the training samples through the split to the bottom nodes of the tree.

Figures 8 and 9 show the distribution of execution time measured in CPU clock ticks and the mean number of clock ticks per instruction, respectively. One can see that operations (1)-(4) add up to about 90% of the execution time (see the above description of operations (1)-(4)). Note a high CPI value for a sampling operation (1) and the corresponding low clock ticks percentage.

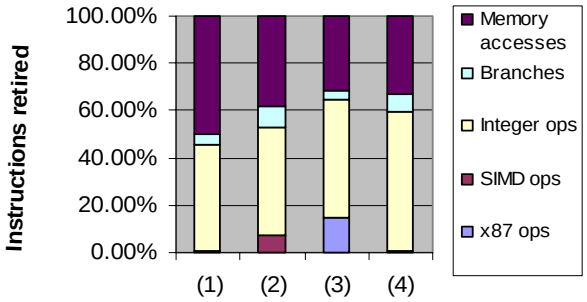


Figure 10: Instruction mix count

One could divide all instructions into four main types: integer and logic operations, floating-point operations, branches, and memory reads/writes. Figure 10 shows the number of retired instructions of each type for (1)-(4). The procedure of sampling is characterized by intensive memory access; sorting requires a large number of comparisons (optimized using SIMD extensions), logical operations and branches, while a search for an optimal split requires intensive FP computations.

Figure 11 illustrates levels of interaction with the memory subsystem. It shows overall memory traffic and its distribution between the caches hierarchy and the main memory (the Xeon processor has 8 KB of L1, 512 KB of L2, and 1 MB of L3 cache). The data explain the high number of clock ticks per retired instructions for (1). The cause is high access rate to the main memory.

Table 1: Characteristics of instructions decoder

	% of decoded commands	Mispredicted branches per instructions retired	Branch prediction rate
(1)	91.697	0.013	78.443
(2)	92.571	0.024	72.964
(3)	95.439	0.002	94.41
(4)	95.322	0.005	94.489

Table 1 characterizes the work of the instruction decoder. It contains the percentage of instructions that did not cause stalls due to the latency of decoding. As the main cause, the number of branches per instructions retired and the BP rate is shown too. As one can see, there is a large number of mispredicted branches in subsampling and sorting procedures that cause stalls of the instruction decoder.

Multithreading

Building a decision tree could be effectively parallelized at the data level. Several algorithms of building tree ensembles (e.g., PEL [11]) assume that each tree is built independently. Then, each sub tree in a decision tree could be built independently, resulting in fine thread granularity. The main thread could make a few first splits, and then assign the building of each sub tree to auxiliary threads. Finally, each of the operations (1)-(3) contains an outer loop on input variables. Iterations are independent, but an aggregation is required at the end (e.g., comparison of split goodness and finding the best one).

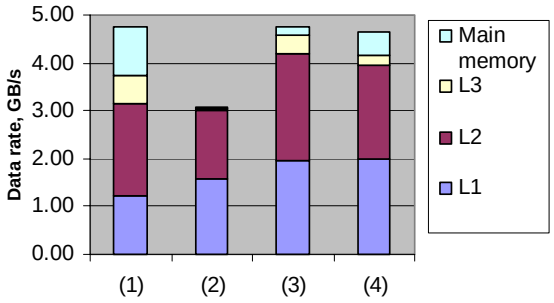


Figure 11: Interaction with memory subsystem

However, access to data structures could be a problem when considering real system architecture. For the cluster-based solution, a subset of training data should be passed to each node, and partitioning is then determined by several first splits. However, the algorithm could be modified to allow effective distributed computations. For SMP systems, the bus becomes a limiting factor with increasing numbers of processors. Figure 12 illustrates this. The vertical axis shows a ratio of execution time per model, related to the execution time of the sequential version. One can see that performance grows near-linearly for up to three threads, and does not grow when the number of threads exceeds six.

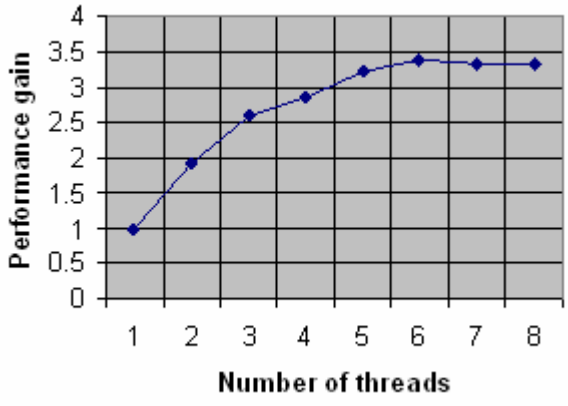


Figure 12: Speedup on the number of threads

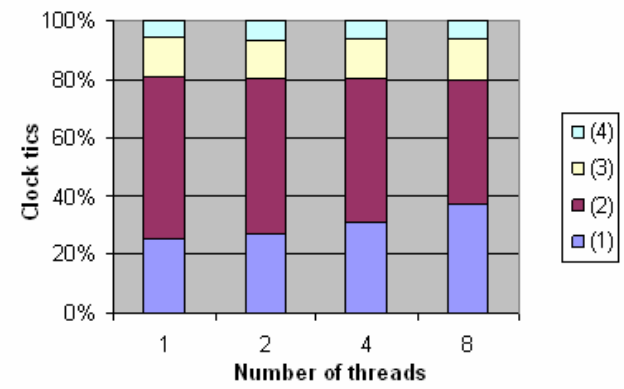


Figure 13: Clock ticks for each function vs. the number of threads

DISCUSSION

The computation of a single tree in a PEL algorithm consists of both intensive data exchange with memory subsystem and CPU load. Function (1) that selects a subset of samples from the original dataset copies a significant amount of data from the source array residing mainly in the main memory to the destination array that is smaller and sits in cache. Figure 11 shows that function (1) causes a significant amount of traffic into main memory. In fact, a significant amount of interaction with cache is caused by a hardware prefetch that reads local data into cache automatically. Still, selecting a subset of samples and variables generates a sparse memory access pattern, and prefetch is not capable of loading all the data we need. The same chart indicates that function (2) works with cache data only. Function (4) divides the training table into right and left according to the currently built split. It generates about a half of the traffic to main memory compared to function (1) but it works more than five times faster. This is because of the difference in memory access patterns: function (4) goes through samples one by one while function (1) jumps between samples and variables in a sparse manner.

Performance gain in Figure 12 shows a sub-linear trend up to three threads. Small gain in Thread 4 is caused by an increased competition between threads for the bus to the main memory. While we do not give a precise interpretation of this effect within this paper, we note that this is in accordance with the data presented in Figure 7. Roughly two-thirds of the execution time in one thread is occupied by the function (2) that is supposed to have excellent scalability. The remaining one-third relates mostly to operations with memory, and this explains the competition that shows up when the number of processors is more than three. The measurements of Figure 11 were taken on a 4-way server with Hyper Threading (HT) enabled. Threads 5-8 share the physical processor with Threads 1-4. The 15-20% gain we get from HT is due to better resource utilization: while one thread waits for the data from memory, executing, for instance, function (1), another thread on the same processor runs calculations of function (2).

Figure 13 supports the hypothesis of poor scalability of function (1): the percentage of time occupied by function (1) grows with the number of threads. If threads have been executed independently from each other, the percentage of clock ticks for each function would have been constant. The data locality issue is crucial here. Figure 8 illustrates the CPI trends for different functions and training samples number. One can see that for a larger dataset it takes more time for function (1) to execute a single instruction, while the CPI rate for function (2) is almost constant.

It is important to note that all data mentioned in the paper have been obtained for the dataset that has sample-wise memory layout. In other words, it is organized in memory so that the data related to one sample occupy a continuous block of memory. (The case of variable-wise representation when the data related to each variable are put into a continuous block of memory lies outside the scope of this paper.) We note, however, that using one representation or another can improve locality and provide an additional speed-up depending on the parameters of the training dataset and PEL.

RESULTS

Learning of tree-based models in the context of large-scale data-mining problems provides many challenges for a computing platform. Often, the more computational power we have the higher prediction power we can get from the model.

The experiments show that an IA-32 system can handle complex ensemble-based learning algorithms very efficiently. The key limiting factor is latency to main memory. A problem-specific data structure can improve data locality and performance gains. For current bus and cache sizes, the algorithm could be effectively parallelized up to four processors. Having several threads per core can provide an additional speed-up.

ACKNOWLEDGMENTS

We thank Roman Belenov and Dmitry Budnikov from Intel Russian Research Center for fruitful discussions.

REFERENCES

[1] Lyman, P. and Hal, R. V., *How Much Information*, 2003 retrieved from http://www.sims.berkeley.edu/how-much-info-2003*.

[2] Spiegelhalter, D. J., Abrams, K., and Myles, J. P., *Bayesian Approaches to Clinical Trials and Health*

*Care Evaluation**, Chichester: John Wiley & Sons, 2004.

[3] Breiman L., Friedman, J.H., Olshen, R.H., and Stone, C.J., *Classification and Regression Trees*, Wadsworth, Belmont, California, 1984.

[4] Breiman, L., Bagging Predictors, *Machine Learning*, vol. 24, no. 2, pp. 123-140, 1996.

[5] Freund, Y. and Schapire, R.E., "Experiments with a New Boosting Algorithm," in *Proceedings of the Thirteenth International Conference on Machine Learning*, pp. 148-156, 1996.

[6] Ho, T. K. Y., "The random subspace method for constructing decision forests," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 20, no. 8, pp. 832-844, 1998.

[7] Breiman, L., "Using adaptive bagging to debias regressions," *Technical Report 547*, Dept. of Statistics, University of California, Berkeley, 1999.

[8] Dietterich, T.G., "Ensemble methods in machine learning," in *Multiple Classier Systems*, Cagliari, Italy, 2000.

[9] Friedman, J. H., *Stochastic gradient boosting*, 1999. http://www-stat.stanford.edu/~jhf/ftp/stobst.ps*

[10] Friedman, J.H., "Greedy Function Approximation: A Gradient Boosting Machine," *Annals of Statistics*, 29, 5, pp. 1189-1232, 2001.

[11] Breiman, L., "Random forests, random features," *Technical Report*, University of California, Berkeley, 1999.

[12] http://www.salford-systems.com/*

[13] Goodwin, R. et al., "Advancements and Applications of Statistical Learning/Data Mining in Semiconductor Manufacturing," *Intel Technology Journal, Volume* 8, *Issue 4*, 2004.

[14] Hastie T., Tibshrirani R., and Friedman J., *The Elements of Statistical Learning*, Springer, New York, 2001.

AUTHORS' BIOGRAPHIES

Alexander Evgenyevich Borisov was born in Nizhny Novgorod, Russia and received his Masters degree in Mathematics (Lie algebras) at Lobachevsky Nizhny Novgorod State University where he is currently working on a Ph.D. degree in the area of context-free grammars. He currently works at Intel (Nizhny Novgorod) as a software engineer and researcher. His technical interests include artificial intelligence and data mining, especially tree-based classifiers. His e-mail is alexander.borisov at intel.com.

Igor Chikalov is a team leader/research engineer in the Analysis & Control Technology department at Intel. He has been working at the R&D site in Nizhny Novgorod (Russia) since 2000. His research interests include machine learning, test theory, statistical modeling. Igor received his Ph.D. degree from Nizhny Novgorod State University in 2002. His e-mail is igor.chikalov at intel.com.

Victor Eruhimov is a senior research scientist in the Intel Russia Research Center. He is leading a team of researchers investigating the computational properties of algorithms in the area of computer vision and machine learning. He was a senior researcher in the Open Computer Vision project, working on the development of the OpenCV library and research in computer vision. Victor received a Masters degree from the Advanced School of General and Applied Physics in the Institute of Applied Physics, Russian Academy of Sciences, in 1999. His e-mail is victor.eruhimov at intel.com.

Eugene Tuv is a staff research scientist in the Analysis & Control Technology department at Intel. His research interests include supervised and unsupervised non-parametric learning with massive heterogeneous data. He holds postgraduate degrees in Mathematics and Applied Statistics. His e-mail is eugene.tuv at intel.com.

Parallel Computing for Large-Scale Optimization Problems: Challenges and Solutions

Mikhail Smelyanskiy, Corporate Technology Group, Intel Corporation
Stephen Skedzielewski, Corporate Technology Group, Intel Corporation
Carole Dulong, Corporate Technology Group, Intel Corporation

Index words: optimization, linear programming, quadratic programming, interior point method, sparse linear system of equations, Cholesky factorization, backward solver, forward solver, elimination tree, supernode, parallel computing, multiprocessor system, shared-memory programming model, message-passing programming model, problem structure, block-angular matrices, asset liability management

ABSTRACT

Optimization refers to the minimization (or maximization) of an objective function of several decision variables that have to satisfy specified constraints. There are many applications of optimization. One example is the portfolio optimization problem where we seek the best way to invest some capital in a set of n assets. The constraints might represent a limit on the budget (i.e., a limit on the total amount to be invested), the requirement that investments are nonnegative (assuming short positions are not allowed), and a minimum acceptable value of expected return for the whole portfolio. The objective or cost function might be a measure of the overall risk or variance of the portfolio return. In this case, the optimization problem corresponds to choosing a portfolio allocation that minimizes risk, among all possible allocations that meet the firm requirements. Another example is production planning and inventory: the problem is to determine the optimal amount to produce in each month so that demand is met while the total cost of production and inventory is maintained without shortages.

In recent years the Interior Point Method (IPM) has became a dominant choice for solving large optimization problems for many scientific, engineering, and commercial applications. Two reasons for the success of the IPM are its good scalability on existing multiprocessor systems with a small number of processors and its potential to deliver a scalable performance on systems with a large number of processors. IPM spends most of its runtime in several important sparse linear algebra kernels. The scalability of these kernels depends on several key factors such as problem size, problem sparsity, and problem structure. This paper describes the computational kernels that are the building blocks of IPM, and we explain the different sources of parallelism in sparse parallel linear solvers, the dominant computation of IPM. We analyze the scalability and performance of two important optimization workloads for solving linear and quadratic programming problems.

INTRODUCTION

Optimization refers to the minimization (or maximization) of an objective function of several decision variables that have to satisfy specified constraints. It enables businesses to make better decisions about how to commit resources, which include equipment, capital, people, vehicles, raw materials, time, and facilities.

While existing hardware performs well on problems with tens of thousands of constraints and hundreds of thousands of variables, it lacks the necessary computational and bandwidth resources to target future datasets whose solution will require teraflops of computation and gigaflops of bandwidth. As an example, consider the Asset Liability Management (ALM) problem from computational finance, where the goal is to coordinate the management of assets and liabilities over several time periods to maximize the return at the end of the final time periods. To hedge against risk requires diversification of a portfolio with many assets; considering more time periods means better planning. Our simple back-of-the-envelope estimate shows that modeling just three time periods and as few as seventy-four assets creates an optimization problem that takes about one hour to solve on today's platforms, but only ten seconds to solve on a teraflop platform of tomorrow.

In order for an optimization workload to achieve high performance on future parallel architectures, one needs to understand (i) the source of parallelism, (ii) how the parallelism changes for different optimization problems, and (iii) how to extract this parallelism for a given problem.

We focus on the Interior Point Method (IPM), a dominant choice for solving large-scale optimization problems in many scientific, engineering, and financial applications. While complex mathematical analysis is the driving force behind IPM, most of the algorithm's computation time is spent in a few sparse linear algebra functions: sparse linear solvers, matrix-matrix multiplication, matrix-vector multiplication, and a few others. Developing parallel systems that efficiently execute these few functions is paramount to high performance for the IPM.

In this paper, we discuss IPM workloads as well as several approaches to parallelizing IPM. First, we discuss important computational kernels that are the building blocks of IPM. Second, we explain several sources of parallelism in sparse parallel linear solvers, the dominant computation of IPM. We also describe how additional parallelism within IPM can be discovered by exploiting inherent problem structures. Thirdly, we present the scalability results and performance analysis of shared-memory IPM on several datasets from linear programming. This workload utilizes highly optimized, parallel routines from the Intel Math Kernel Library, built using the PCx framework and parallelized by our team. We also present scalability results and performance analysis of a structure-exploiting quadratic IPM workload, the Object-Oriented Parallel interior point Solver (OOPS), on asset liability and management problems.

OPTIMIZATION AND THEIR USAGE MODELS

An optimization problem, has the form

$$\text{minimize } f_0(x)$$

$$\text{subject to } f_i(x) \quad b_i, \; i = 1,\ldots, m$$

Here the vector $x = (x_1,\ldots,x_n)$ is the optimization decision variable of the problem, the function $f_0(x)$ is the objective function, the functions f_i , $i = 1,\ldots,m$, are the (inequality) constraint functions, and the constants $b_1,\ldots,b_m$ are the limits, or bounds, for the constraints. A vector x^* is called optimal, or a solution of the optimization problem if it has the smallest objective value among all vectors that satisfy the constraints.

There are several important classes of optimization problems, characterized by particular forms of the objective and constraint functions. As an example, the optimization problem is called a Linear Program (LP) if the objective and constraint functions f0,…, fm are linear functions of x. The LP optimization problem is of the form

$$\min c^T x, \text{ subject to } Ax=b, \; x \quad 0$$

where A is m by n the matrix of linear constraints, and vectors x, c, and b have appropriate dimensions.

Another important example, the convex quadratic optimization problem (QP), is of the form

$$\min c^T x + \tfrac{1}{2} x^T Q x, \text{ subject to } Ax=b, \; x \quad 0$$

where Q is n by n positive semidefinite matrix, and A, x, c, and b are the same as in LP.

LP and QP are important not only because many problems encountered in practice can be formulated as either LP and QP problems, but also because many methods for solving general non-linear programming problems (NLP) solve them by solving the sequence of linear (sequential linear programming) or quadratic (sequential quadratic programming) approximations of the original NLP problem.

There are many applications of optimization. In the radiation therapy planning optimization problem, when choosing a plan for any individual patient, one seeks to determine radiation beam directions and intensity with the goals of maximizing the delivered dose to the tumor while minimizing the dose in normal tissue and organs at risk. There exist different formulations of this problem as LP, QP, or NLP.

In production planning and inventory problems, the problem is to determine the optimal amount to produce in each month so that demand is met yet the total cost of production and inventory is minimized and shortages are not permitted. This problem has been traditionally solved using the LP approach.

Another example is the famous portfolio optimization problem where we seek the best way to invest some capital in a set of n assets. The variable x_i represents the investment in the *i*th asset, so the vector $x=(x_1,\ldots,x_n)$ describes the overall portfolio allocation across the set of assets. The constraints might represent a limit on the budget (i.e., a limit on the total amount to be invested), the requirement that investments are nonnegative (assuming short positions are not allowed), and a minimum acceptable value of expected return for the whole portfolio. The objective or cost function might be a measure of the overall risk or variance of the portfolio return. In this case, the optimization problem corresponds to choosing a portfolio allocation that minimizes risk, among all possible allocations that meet

the firm requirements. The problem is known as the Markovitz mean-variance optimization problem and is modeled using QP.

The last example is device sizing in electronic design, which is the task of choosing the width and length of each device in an electronic circuit. Here the variables represent the widths and lengths of the devices. The constraints represent a variety of engineering requirements, such as limits on the device sizes imposed by the manufacturing process, timing requirements that ensure that the circuit can operate reliably at a specified speed, and a limit on the total area of the circuit. A common objective in a device sizing problem is the total power consumed by the circuit. The optimization problem is to find the device sizes that satisfy the design requirements (on manufacturability, timing, and area) and are most power efficient. This problem can be modeled using LP or QP.

INTERIOR-POINT METHOD (IPM)

In the past decade, the IPM has become a method of choice for solving large convex optimization problems. As parallel processing hardware continues to make its way into mainstream computing, it becomes important to investigate whether parallel computation can improve the performance of this commercially vital application.

The IPM has a unified framework for LP, QP, and NLP. The method starts with the initial guess to the solution of the optimization problem, *x*. The core of the method is the main optimization loop, which updates the vector x at each iteration until the convergence to the optimal solution vector x* is achieved. A key to efficient implementation and parallelization of IPM is that all three algorithms depend on four linear algebra kernels listed below:

1. Form linear systems of equations, Mx=b, where M is the symmetric matrix of the form $M = \begin{bmatrix} -Z & A^T \\ A & 0 \end{bmatrix}$, where A is the original matrix of constraints. The matrix of this form is called *augmented* system. For linear programming problems, where Z is a diagonal matrix, one uses substitution of variables in the above linear system of equations, so that matrix M is reduced to *normal* equation form $M = AZ^{-1}A$. This requires a **matrix-matrix multiplication** operation.
2. **Cholesky factorization** of matrix M = L D L^T in order to solve the system of linear equations, Mx=b. Here L is lower triangular, D is diagonal if M is positive definite, and D contains 1 by 1 and 2 by 2 bocks if M is indefinite. This step is normally the most time-consuming step of the IPM.
3. **Triangular solver** uses result of factorization to solve a system of linear equations (L D L^T)x=b, using the following three steps
    a. Forward solver, solves Ly=b
    b. Diagonal solver solves Dz=y. Note that when normal equations are used in the case of LP, this step can be eliminated, because the diagonal matrix D is positive and hence M can be represented as M=(L') $(L')^T$, where L' = L $D^{1/2}$.
    c. Backward solver solves L^Tx=z
4. **Matrix vector multiply**: Ax, A^Tx (transpose matrix vector multiply), and Mx (symmetric matrix-vector multiply).

Other operations, such as inner products, vector additions, and vector norm computation contribute a small amount compared to the above operations.

We see that the parallel efficiency of IPM depends on the efficient parallel implementation of these four linear algebra kernels. For the majority of realistic problems, solving systems of equations (kernels 2 and 3) is the most time-consuming portion of the IPM. For most optimization dataset models, the underlying matrix M is very sparse. As will be explained in later sections, sparsity is important because it uncovers an additional coarse-level parallelism, which is otherwise unavailable in the dense problems.

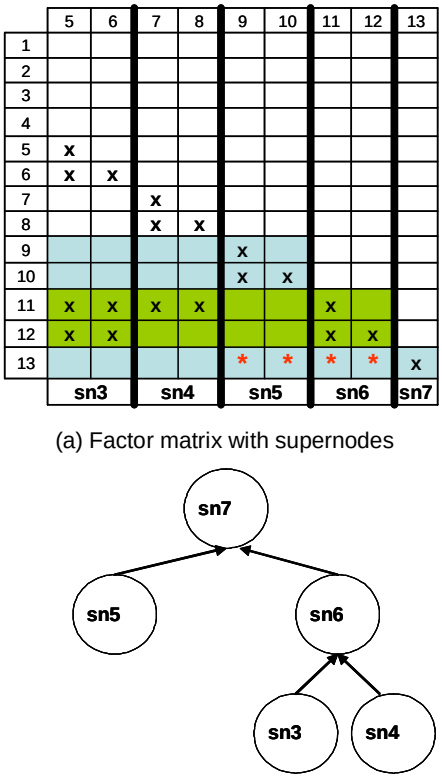
PARALLELIZATION OF IPM

In this section we describe the serial and parallel algorithm for solving sparse linear systems of equation. We discuss different levels of parallelism that can be explored for unstructured problems as well as additional levels of parallelism that become available in structured problems.

Sparse Unstructured Problems

This section deals with sparse unstructured matrices that arise from general optimization LP, QP, and NLP problems. Such matrices possess no distinct and exploitable non-zero structure. In the rest of this paper, we assume that the original symmetric matrix M is scattered into the factor matrix L. Two fundamental concepts behind solving sparse systems of linear equations are ***supernode*** and ***elimination tree***. A supernode is a set of contiguous columns in the factor L whose non-zero structure consists of a dense triangular block on the diagonal and an identical set of non-zeroes

for each column below the diagonal. An example of supernode is given in Figure 1(a).



(a) Factor matrix with supernodes

(b) Elimination Tree

Figure 1: Example of factor matrix L supernode and its elimination tree

Figure 1(a) shows 13x13 sparse matrix L: empty entries are assumed to have zero values. In this example there are six supernodes. For example, columns 1 and 2 form supernode sn1, columns 7 and 8 form supernode sn4, and columns 11, 12, and 13 form supernode sn6. Since all columns in the supernode have an identical non-zero structure, in practice non-zero elements of supernodes are compactly compressed into and stored as a dense matrix.

The elimination tree is a task dependence graph that characterizes the computation and data flow among the supernodes of L during Cholesky and triangular solve, and it is defined as follows: *parent(sn_j) = min{sn_i | i > j and at least one of the elements of sn_j which correspond to the diagonal block of sn_i is non-zero}*. In other words, the parent of supernode j is determined by the first sub-diagonal non-zero in supernode i. Figure 1(b) shows an example of the elimination tree for the matrix in Figure 1(a). We see that there is an edge between sn1 and sn5, because as the shaded portion of the figure shows, the second row of the 2 by 2 diagonal block of sn5 corresponds to the non-zero row 10 in sn1. Similarly, there is an edge between sn3 and sn6, because the first two rows of sn6 correspond to non-zero elements in rows 11 and 12 of supernode 3.

Given the elimination tree, the Cholesky factorization forward and backward kernels can all be expressed using the following generic formulation:

T = breadth-first traversal of ET (bottom-up or top-down)
for each supernode sn_i in ET
    perform processing task on sn_i
endfor

The order of the tree traversal and the processing task are different for each kernel. Cholesky and the forward solver perform bottom-up traversal of the elimination tree, whereas backward solver performs top-down traversal of the elimination tree. The processing task for forward and backward solver are very similar; however, we do not discuss them here due to space limitations. The details of a Cholesky processing task are discussed next.

Cholesky Factorization

Cholesky factorization is the most time-consuming operation among the four kernels. According to our experiments (see our experimental section results for unstructured problems) on average, IPM spends 70% of the time in this kernel. The high-level pseudo-code of Cholesky is given in Figure 2(a). Cholesky processing task (Lines 3-7) is generally expressed in terms of two primitive operations on the supernode, **cdiv** and **cmod**, both of which are shown in Figure 2(b) and Figure 2(c), respectively.

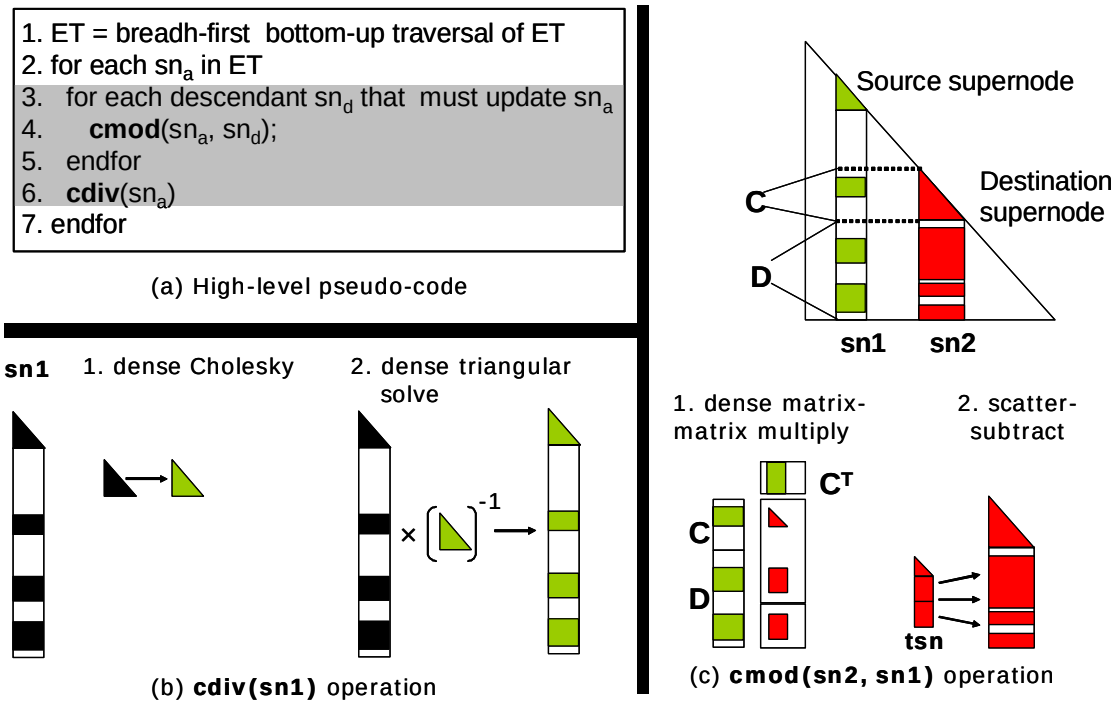


Figure 2: Cholesky factorization

Given supernode sn1, cdiv(sn1), also known as *supernode factorization*, requires multiplication of the dense rectangular portion of the supernode below its main diagonal by the inverse of the supernode's dense diagonal block. The inversion is not actually computed. Rather, this computation is broken into two steps shown in Figure 2(b) for supernode sn1. In the first step, we perform dense Cholesky on the diagonal block of sn1. In the second step, we solve a large number of triangular systems for each nonzero row of the supernode below the main diagonal.

The second and most time-consuming primitive operation in Cholesky factorization is called cmod, which is also known as *supernode-supernode update*. cmod(sn2, sn1) operation adds into destination supernode sn2 the multiple of the source supernode sn1 and, similar to cdiv, consists of two steps, shown in Figure 2(c). The first step uses dense matrix-matrix multiply, to multiply the sn1 by the transpose of its sub-matrix C which corresponds to the dense triangular block of the sn2. This results in the temporary supernode tsn. In the second step, the temporary supernode tsn is scatter-subtracted from the second supernode sn2. The scatter operation is required because tsn and sn2 may have different non-zero structures (which is the case in Figure 2(c)).

The high-level pseudo-code of Cholesky, shown in Figure 2(a), scatters original matrix M into the factor L and performs a series of cdiv and cmod operations on supernodes of L to factorize it. The algorithm performs breadth-first bottom-up traversal of the elimination tree (designated as ET in the figure) starting from the leaves (Line 2). Each supernode sn_a, receives cmod updates from its descendant supernodes sn_d (Lines 3-6). Upon receiving all the updates, a cdiv operation is performed on sn_a to complete factorization of the supernode. The factorized supernode is now ready to update its own ancestors. All ancestor supernodes that require an update from the descendent supernode are known in advance; the leave supernodes require no updates. Note that due to the fact that each supernode is stored as a dense matrix, one can use efficient implementations of dense linear algebra subroutines (such as BLAS 2, BLAS 3, and LINPACK) to perform cdiv and cmod operations.

Understanding Parallelism in Cholesky Factorization

We now examine the opportunities for parallelism in the above implementation of sparse Cholesky in Figure 2(a). This implementation contains parallelism on several different levels:

Level 1: Elimination tree parallelism, which corresponds to the parallel execution of the outermost loop in Lines 2-7. Here several iterations of the loop, which

correspond to independent sub-trees of the elimination tree, can be started in parallel on several processors. In other words, if T1 and T2 are disjoint sub-trees of the elimination tree, with neither root node a descendant of the other, then all of the supernodes of T1 can be factorized completely independently of the supernodes corresponding to T2, and vice versa. Hence, these computations can be done simultaneously by separate processors with no communication between them.

Level 2: The second level of parallelism exists in the innermost loop (Lines 3-5) and is essentially a parallel reduction operation. For a given ancestor supernode sn_a, all updates from its descendents (Line 4) can proceed in parallel. Note, however, that it may happen (more often than not) that two or more descendents will try to scatter-subtract into the same elements of their ancestor. This requires some locking mechanism to guarantee that only one update happens at a time.

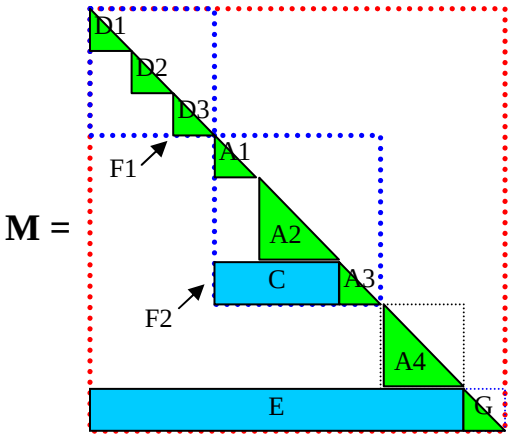
Level 3: The third level of parallelism exists within an individual cmod update operation called from the innermost loop (Line 4). Due to the fact that each cmod operation is composed of dense Cholesky and dense matrix-matrix product operation, each can be further parallelized. The parallelism also exists within the cdiv operation performed on a given supernode sn. As explained earlier, the cdiv(sn) operation involves solving a large independent set of dense triangular systems; hence all such solves can be done in parallel.

To quantify the parallelism inherent to sparse Cholesky factorization, we note that the number of operations required to factorize a typical sparse matrix on a single processor is roughly $\Theta(nnz^{3/2})$, where nnz is a number of non-zeros in the matrix. Assuming only one column per supernode, unlimited hardware resource and zero-communication cost, at each step of parallel computation we will execute as many parallel factorization operations as possible, constrained only by the data-dependencies within the elimination tree, which are inherent to a particular sparse dataset. Therefore the number of parallel steps to factorize the sparse matrix is a critical path through the elimination tree. The height of a well-balanced elimination tree is approximately $\Theta(\ln(N))$, where N is the number of row/columns of M. Thus, the expected ideal speed-up of Cholesky is approximately $\Theta(nnz^{3/2}/\ln(N))$.

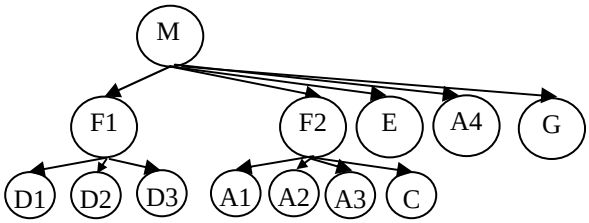
Sparse Structured Problems

Of the four levels of parallelism described above, the elimination-tree-level parallelism is the coarsest and therefore is the most attractive to exploit with parallel processing. However, to exploit this parallelism efficiently requires a balanced elimination tree. Different elimination trees can be constructed for a given symmetric matrix M when the matrix is re-ordered using symmetric row and column permutations. Obviously an elimination tree where all sub-trees have a similar height will result in better parallel speed-up than one where most of the nodes are in one long branch. However, finding a re-ordering of the matrix that leads to a more balanced elimination tree is a non-trivial task (in fact it is NP-complete).

In many situations, however, explicitly constructing a balanced elimination tree is not necessary. Many truly large-scale optimization problems are not only sparse but also display some flavor of block structure that make them highly amenable to parallelism [4]. By a block-structured matrix we understand a matrix that is composed of sub-matrices. A block-structured matrix can be nested, where each sub-matrix is a block-structured matrix itself. The example of the nested block-angular matrix is given in Figure 3(a).



(a) Example of block-angular matrix



(b) Tree representation

Figure 3: Nested block structured matrix and its tree representation

The nested block-structure of a matrix can be thought of as a tree. Its root is the whole matrix, and every block of a particular sub-matrix is a child node of the node representing this sub-matrix. Leaf nodes correspond to the elementary sub-matrices that can no longer be divided into blocks. Figure 3(b) shows an example of the

matrix M's tree. We see that sub-matrix F1 of M has a diagonal structure, whereas sub-matrix F2 of M has a block-angular structure.

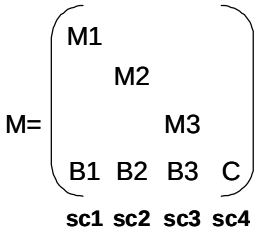
The existence of a well-defined structure is due to the fact that many optimization problems are usually generated by a process involving discretization of space or time (such as control problems or other problems involving differential equations), or of probability distribution (such as stochastic programming). Other sources of structure are possible such as in a network survivability problem where slight variations of the core network matrix are repeated many times. Note that a block of a block-structured matrix can itself be a sparse matrix. Therefore block structure offers the additional coarse level of parallelism, which is coarser than the elimination tree level of parallelism.

Structure-Oriented Cholesky Factorization

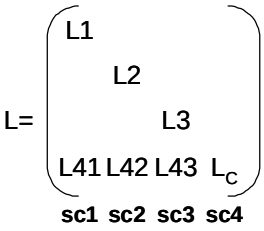
The efficient exploitation of matrix structure by linear algebra routines is based on the fact that any method supported by the linear algebra library can be performed by working through the tree: at every node, evaluating the required linear algebra operation for the matrix corresponding to this node can be broken down into child nodes in the tree. Different structures, however, need their own linear algebra implementation.

In Figure 4(a), we show by means of a simple example how Cholesky factorization can be implemented on block-angular matrix M. This matrix consists of 4 by 4 blocks. It is easy to see that the factor matrix L shown in Figure 4(b) is also block-angular. We partition L into 4-column blocks: sc_i consists of sub-blocks L_i and L_4i for i=1,2,3, and sc4 contains a single block L_C. Note that the column blocks are similar to supernodes for unstructured matrices. The only difference is that a supernode, by definition, must only contain sub-blocks of dense rows, whereas column blocks of a block-structured matrix can contain sub-blocks with an arbitrary sparsity pattern.

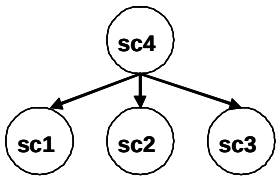
Despite this difference, the same concepts that applied to supernodes equally apply to column blocks. Figure 4(c) shows the elimination for matrix L, and Figure 4(d) shows the application of Cholesky factorization from Figure 2(a) to the column blocks of L. Step 1 is composed of two sub-steps: (i) cdiv(sc1) performs the required operations on sub-blocks L_1 and L_{14},. and (ii) cmod(sc4, sc1) updates L_C of sc4 with the corresponding product of L_{41} L_{41}^T. Note that sc1 only has to update sc4, since, as indicated by the elimination tree, sc4 is its only parent. Similar operations are performed on sc2 and sc3 in steps 2 and 3, respectively. Finally, the cdiv operation is performed on sc4 to factorize the sub-block L_C and to complete the Cholesky factorization of M.



(a) Block-angular matrix M



(b) Factor L of M



(c) Elimination Tree of L

Serial Steps	cdiv(sci)	cmod(scj, sci)
1.	cdiv(sc1): L1=Cholesky(M1), L41 = B1 $L1^{-T}$	cmod(sc4, sc1): $C_{tmp} = C - L41\ L41^T$
2.	cdiv(sc2): L2=Cholesky(M2), L42 = B2 $L2^{-T}$	cmod(sc4, sc2): $C_{tmp} = C_{tmp} - L42\ L42^T$
3.	cdiv(sc3): L3=Cholesky(M3), L43 = B3 $L3^{-T}$	cmod(sc4, sc3): $C_{tmp} = C_{tmp} - L43\ L43^T$
4.	cdiv(sc4): L_C=Cholesky(C_{tmp})	-

(d) Serial computation to factorize M

Figure 4: Exploiting block-angular matrix structure

Understanding Parallelism in Structure-Oriented Cholesky

The serial factorization algorithm in Figure 4(d) lends itself naturally to parallelization as shown in Figure 5. Steps 1, 2, and 3, which are completely independent, get distributed among the three processors, P1, P2, and P3. The resulting computation happens in Phase 1. Notice that instead of simultaneously updating the same matrix C_{tmp}, each processor P_i, stores the result of this update into its private copy of C_{tmp}, C_i. In Phase 2, the global reduction operation is performed, wherein each processor adds its own contribution to C_{tmp}. Finally, in

Phase 3, matrix C_{tmp} is factorized and the result is stored in L_C, which completes parallel factorization of M. Note that the reduction operation in Step 2 can be parallelized. In addition, work performed in Phase 1 on each processor, as well Cholesky factorization in Phase 3, can also be parallelized. If M1, M2, M3, or C_{tmp} is unstructured, the parallel algorithm described in the previous section can be used.

		Phase 1		Phase 2	Phase 3
		Independent Parallel Computation		(Parallel) Reduction	(Parallel) Cholesky
Parallel Steps	P1	L1=Cholesky(M1):L41 = B1 $L1^{-T}$	C1= L41 $L41^T$	C_{tmp}=C– C1–C2–C3	L_C=Cholesky(C_{tmp})
	P2	L2=Cholesky(M2):L42 = B2 $L2^{-T}$	C2=L42 $L42^T$		
	P3	L3=Cholesky(M3):L43 = B3 $L3^{-T}$	C3=L43 $L43^T$		

Figure 5: Split of computations between processors in structure-oriented Cholesky

Hence we see that by exploiting the special structure of the matrix, we are able to exploit the coarser level of parallelism unlike in cases of unstructured matrices.

PERFORMANCE ANALYSIS OF LINEAR AND QUADRATIC IPM WORKLOADS

In this section we present the scalability results and performance analysis of two applications. The first application is the shared-memory implementation of IPM for solving arbitrary unstructured linear programming problems. The second application is the MPI implementation of a structure-exploiting quadratic IPM workload (OOPS) for solving structured asset liability management quadratic programming problems. We performed both experiments on a 4-way 3.0 GHz Intel® Xeon™ processor MP-based system, with 8 GB of global shared memory and three levels of cache on each processor: 16 KB L1, 512 KB L2, and 4 MB L3. The four processors and memory are connected with a ServerWorks GC-HE, capable of delivering the peak bandwidth of 6.4 Gb/s.

Performance Characterization of IPM for Unstructured Linear Programs

For these experiments we use an Interior Point Solver (IPS) workload [3], which our team built for solving linear programming problems. IPS is based on PCx, a serial interior-point linear programming package developed at Argonne National Laboratory in collaboration with Northwestern University [5]. Our implementation of the Cholesky factorization and the solver routines uses a parallel sparse direct solver package, called PARDISO, developed at the University of Basel [6], which is now included as part of Intel's Math Kernel Library. In addition, we implemented and parallelized the routines for sparse matrix-matrix multiplication and sparse matrix-vector multiplication. Note that both the PARDISO code and our routines are parallelized using OpenMP.

Table 1 summarizes the statistics for the datasets used in our experiments. They mostly come from the standard NETLIB test set and represent realistic linear models from several application domains. Columns 1 and 2 show the number of variables and constraints in the constraint matrix for each problem. Column 3 shows the size of M, and Column 4 shows the number of non-zeros in its factor L. Note that the datasets are sorted in the order of increasing number of non-zeros. The last column shows the density of the factor matrix, computed as 100%*non-zeros/(neqns^2). We see that on average, the problems are fairly large and most of them are very sparse.

® Intel and Xeon are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States and other countries.

Table 1: Characteristics of LP datasets

	nconstraints	nvariables	neqns	nlns	Density (%)
ken-18.dat	78862	128434	78862	2175306	0.034977
fleet12.dat	21616	67841	21616	4085152	0.874294
pds-20.dat	32287	106180	32287	6388010	0.612788
fome13.dat	47872	97144	47872	10668201	0.465509
snp30lp1	297998	953120	297998	20352321	0.022919
gismondy.dat	18262	23266	18262	30887926	9.261729

Figure 6(a) shows the breakdown of total execution time spent in the main optimization loop into the four important parallel regions and the remaining serial region. The parallel regions are Cholesky factorization, triangular solver (both forward and backward), matrix-matrix multiply (mmm), and matrix-vector multiply (mvm). For each dataset, we show four bars corresponding to one (1P), two (2P), and four (4P) processors, respectively. Each bar is broken into five parts, one for each execution region. Note all the times are relative to one processor runtime, and the number on the top of one processor shows the total time spent in the main optimization loop. We see that for many datasets Cholesky is the most time-consuming kernel (main optimization loop of IPM spends on average 70% factorizing the matrix), and it also achieves good scalability for these datasets. The solver, which is the second most-time consuming kernel (17% of time on average), scales worse compared to the Cholesky and will require a considerable tuning effort in order for IPM to scale well on larger numbers of processors. Another 4% of time is spent in parallel mmm, which scales very well up to four processors. An additional 4% of the time is spent in different flavors of parallel mvm. The remaining 5% of the time is spent in the serial region.

Figure 6(b) reports the speed-up of IPS on one, two, and four processors. The highest speed-up (2.7x on four processors) is attained for the gismondi dataset, which is also the largest dataset in terms of the number of non-zero elements. Comparing Table 1 and Figure 6(b), we see that both the run-time and scalability of IMS are almost perfectly correlated with the number of non-zeros in the factor matrix that represents the problem size. This is encouraging as it suggests that parallel computation improves the performance on harder problems.

We used the Intel Thread Profiler, a parallel performance analysis tool, to identify and locate bottlenecks that are limiting the parallel speed-up of IPS. In this paper we only present the results for Cholesky factorization. The Thread Profiler identifies three important factors that adversely impact speed-up:

1. *Load imbalance* is the time the threads that completed execution wait at a barrier at the end of the parallel region until all remaining threads have completed the assigned work of the region. When unequal amounts of computation are assigned to threads, threads with less work to execute sit idle at the region barrier until those threads with more work have finished.
2. *Locks* is the time a thread spends waiting to acquire a lock.
3. *OpenMP overhead* is the time spent inside the OpenMP Runtime Engine that implements OpenMP.

For all datasets, Figure 7 shows the total Cholesky factorization time (summed over all processors) spent executing instructions (busy time), waiting on acquiring the locks (locks time), waiting on barriers due to load imbalance (imbalance time), and the time spent inside the OpenMP engine (OpenMP overhead time). Note that all results on one, two, and four processors are normalized to the one processor time. Perfect speed-up is possible only when the total time does not increase as the processors increase. As expected from Figure 6(b), ken-18 has the worst speed-up because the busy, wait, and OpenMP times increase by 150% for two processors and by as much as 350% for four processors. We can also observe a 30-60% increase in busy time on two and four processors for the other five datasets. By looking at the source code we identified the cause of these increases: they are due to the busy waiting loop inside the PARDISO implementation of Cholesky, wherein several processors try to acquire a shared lock in order to enter a critical region. The modest scalability of the triangular solver is due to the same reasons. The future implementation of the sparse linear solver within PARDISO is likely to address this issue.

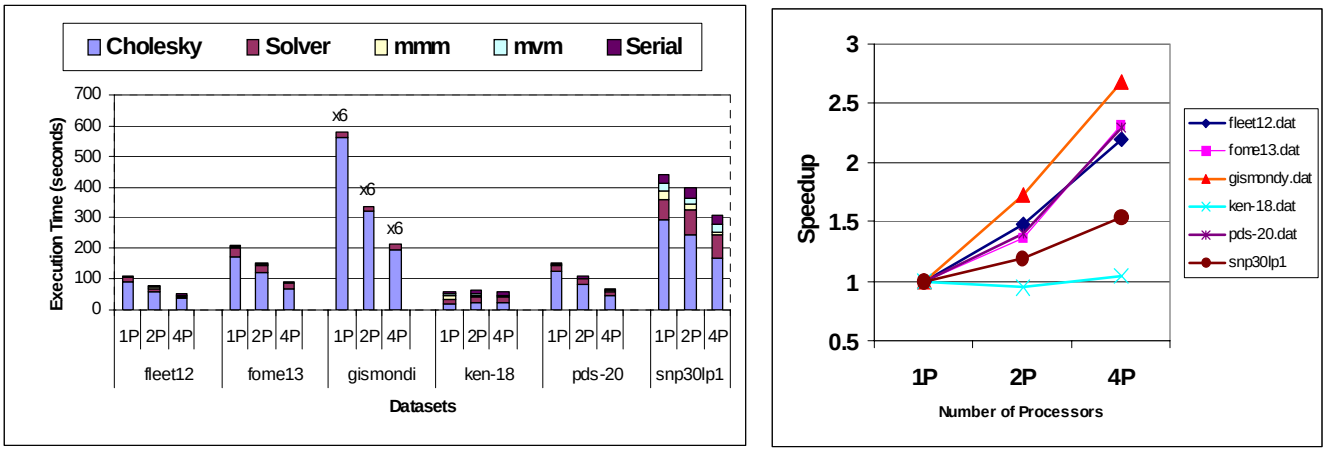


Figure 6: Parallel performance of main optimization loop of IPS

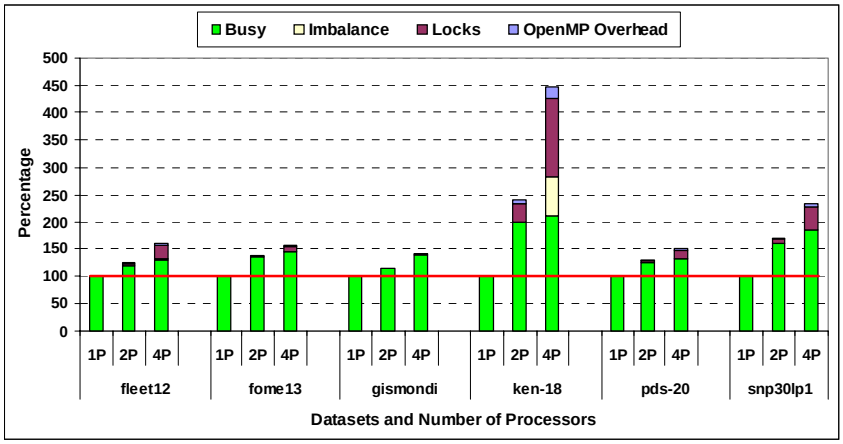


Figure 7: Concurrency and load balance in the parallel Cholesky factorization

Performance Characterization of IPM for the Structured Quadratic Program

For these experiments we used the OOPS workload [7], which we obtained from researchers at the University of Edinburgh. This uses a quadratic programming variant of IPM to solve the ALM problem. ALM is the process of finding an optimal solution to the problem of minimizing the risk of investments whose returns are uncertain. The method associates a risk probability to each asset and uses discrete random events observed at times t = 0, …, T to create a branching scenario tree rooted at the initial time. At each time step the probability of reaching a given node is computed by looking at its predecessor nodes. At the end of the process (time T+1) we can assign a probability to each outcome and compute the asset value at that time. The probability at the leaves of this branching tree will sum to one and we can assess the risk by looking at the asset value vs. the probability graph. The above steps are formulated as a structured quadratic problem with a block-angular structure, which is solved using OOPS.

The research team at the University of Edinburgh also provided problem sets that are summarized in Table 2. Columns 1, 2, and 3 show the number of time steps, the blocked matrices that compose the problem, and the number of assets. The last five columns are the same as given in the linear optimization. Again, we see that these problems are fairly large and very sparse.

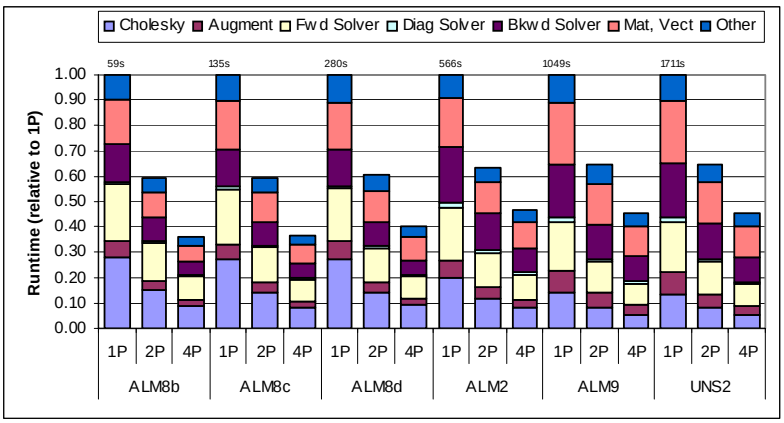
Table 2: Characteristics of ALM datasets

	steps	blocks	assets	nconstraints	nvariables	neqns	non-zeros	Density (%)
ALM8b	3	33	50	57,274	168,451	57,274	1,009,800	0.03%
ALM8c	3	50	50	130,102	382,651	130,102	3.378,750	0.02%
ALM8d	3	70	50	253,522	745,651	253,522	9,070,250	0.01%
ALM2	6	10	5	666,667	1,666,666	666,667	3,611,075	0.08%
ALM9	5	24	4	2,077,207	5,193,016	2,077,207	23,368,500	0.01%
UNS2	109	40	40	2,160,919	5,402,296	2,160,919	27,071,115	0.00%

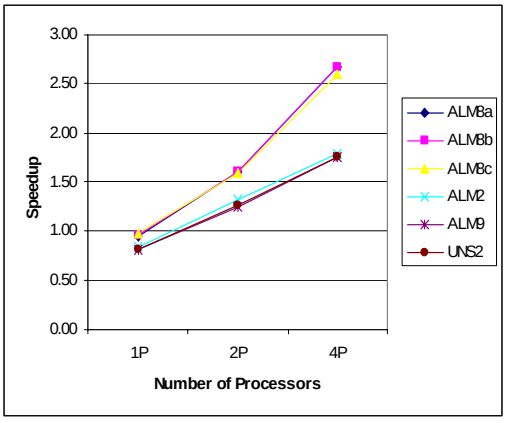
These inputs were run on the same 4-way 3.0 GHz Intel Xeon processor MP-based system that we described earlier. Figure 8(a) shows the breakdown of total execution time for OOPS. The regions appear to be slightly different than in the case of unstructured LP. As explained in the Interior Point Method section, OOPS builds an augmented matrix in each iteration of the optimization loop, whereas IPS performs mmm to form the normal matrix. This matrix is factorized using structure-exploiting Cholesky. Triangular solvers are similar to IPS, but many calls to operations on vectors and matrices are combined into the "Mat, Vect" region. For each dataset, we show four bars corresponding to one (1P), two (2P), and four (4P) processors, respectively. The time shown in this graph is relative to the time taken on the one-processor run. The total time (in seconds) for a one-processor run is given above its bar. The factorization routine has a large parallel section, followed by a global reduction (serial), followed by the redundant Cholesky factorization, which is duplicated in each processor (as described above). This duplication minimizes the communication, but causes the factorization step to exhibit less than linear scalability, as shown in our measurements. A similar pattern (parallel, serial, duplicate-parallel) occurs in the forward and backward solver routines, and we see a similar speed-up as in the factorization step. Since the solver takes a larger fraction of time in OOPS than in IPS, we broke it down into its components. The Mat, Vect section scales in a similar manner to the other routines. These routines have not been heavily optimized and have headroom for additional improvement. We also see a significant amount of overhead (11-13%) on the one-processor run on the larger data sets when compared with a serial version of OOPS. We speculate that this overhead is due to shared memory implementation of MPI, and we plan to investigate the cause of this overhead in our future work.

Figure 8(b) reports the speed-up of IPS for the test datasets on one, two, and four processors. The scalability appears to be correlated with the amount of work required to factor the constraint matrix M. The scaling of OOPS does not yet exploit all of the parallelism that is present in the algorithm.

To understand the performance overhead of the MPI calls, Figure 9 shows the results from running the Intel trace analysis tools on this workload. It instruments the code and measures the time waiting for messages. The instrumented runs show that a relatively small amount of time is spent in the MPI libraries and that almost all of that time is in the MPI reduction routine. It corresponds most closely to the "imbalance" portion of the OMP breakdowns. The rest of the additional time is spent in the OOPS code. We are investigating the source of this extra time.



(a) Execution time breakdown



(b) Speedup

Figure 8: Parallel performance of main optimization loop of OOPS

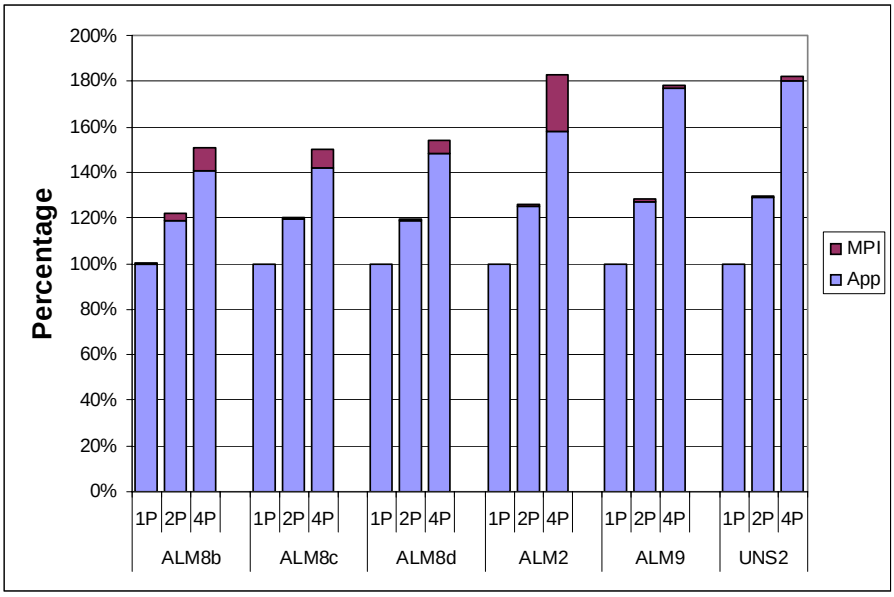


Figure 9: Concurrency and load balance in OOPS

CONCLUSION

In this paper we described a parallel IPM for solving optimization problems. The performance of IPM depends on several key sparse linear algebra kernels. The most important kernel is the solution of the sparse linear system of equations. We described serial and parallel implementations of the sparse linear solver for both unstructured and structured optimization problems.

We have done performance and scalability analysis of IPS–a linear optimization workload for solving unstructured linear programs. We reported up to 2.7x speed-up on the 4-way 3.0 GHz Intel Xeon processor MP-based system for a diverse set of linear problems. We also presented the performance and scalability analysis of OOPS–a structure-exploiting quadratic optimization workload for solving structured quadratic problems. OOPS exposes parallelism by passing structure information from the high-level optimization problems into the linear algebra layer. We achieved up to a 2.7x speed-up on the number of datasets from important asset liability management problems.

Overall, we observed that the scalability of IPM depends on several key factors such as problem size, problem sparsity, as well as problem structure. Although we observed similar performance scalability for the linear unstructured problems and the quadratic structured problems, the structured problems exhibit multiple levels

of parallelism that are not all exploited in the current OOPS implementation. This leaves headroom for performance scalability on systems with large numbers of processors, which we are going to explore in our future work.

One expects the optimization problem size to grow in the future. For example, an increased number of assets in an investor's portfolio will lead to better risk diversification and hence higher return on investment. Many truly large-scale optimization problems are not only sparse but also display block-structure, because these problems are usually generated by discretizations of space or time. These large optimization problems will clearly benefit from a system capable of exploiting multiple levels of parallelism from fine grain to coarse grain.

ACKNOWLEDGMENTS

We acknowledge Radek Grzeszczuk for the key role that he played in the effort to identify and understand Interior Point Method as an important optimization workload. Our thanks go to Jacek Gondzio and Andreas Grothey from the University of Edinburgh for providing the OOPS code and assisting us in understanding its functionality. We also acknowledge Bruce Greer for helping us to acquire and understand the PARDISO solver source code. Finally, we thank Dmitry Ragozin from Intel for his help in understanding the performance bottlenecks of the PARDISO solver.

REFERENCES

[1] J. Nocedal and S.J. Wright, *Numerical Optimization*, Springer-Verlag, New York, Inc, 1999.

[2] S. Boyd and L. Vandenberghe, *Convex Optimization*, Cambridge University Press, 2003.

[3] Pranay Koka, Taeweon Suh, Mikhail Smelyanskiy, Radek Grzeszczuk, and Carole Dulong, "Construction and Performance Characterization of Parallel Interior Point Solver on 4-way Intel Itanium Multiprocessor System," *IEEE 7th Annual Workshop on Workload Characterization (WWC-7*)*, October 2004.

[4] Gondzio, J. and A. Grothey, "Exploiting Structure in Parallel Implementation of Interior Point Methods for Optimization," *Technical Report MS-04-004*, School of Mathematics, The University of Edinburgh, December 18, 2004.

[5] J. Czyzyk, S. Mehrotra, and S. J. Wright, "PCx User Guide," *Technical Report OTC 96/01*, Optimization Technology Center at Argonne National Lab and Northwestern University, May 1996.

[6] University of Basel, PARDISO Direct Sparse Solver. http://www.computational.unibas.ch/cs/scicomp*.

[7] Gondzio, J. and A. Grothey, "Parallel Interior Point Solver for Structured Quadratic Programs: Application to Financial Planning Problems," *Technical Report MS-03-001*, School of Mathematics, The University of Edinburgh, April 16, 2003, revised in December 12, 2003.

AUTHORS' BIOGRAPHIES

Mikhail Smelyanskiy is a member of the Research Staff in the Corporate Technology Group. He received his B.Sc., M.Sc., and Ph.D. degrees in Electrical Engineering and Computer Science from the University of Michigan, Ann Arbor in 1996, 1999, and 2004, respectively. Since he joined Intel in September 2003, he has worked on future multi-core computer architecture to efficiently execute applications from the area of optimization and finance. His graduate work was on VLIW compiler scheduling algorithms for efficient resource utilization. His e-mail is Mikhail.Smelyanskiy at intel.com.

Stephen Skedzielewski is a senior researcher in the Workload Analysis Dept. in the Corporate Technology Group. He recently joined this group after spending nine years analyzing IPF compiler performance. His main technical interests are in performance analysis and parallel computing. He has a B.S. degree from Caltech and M.S. and Ph.D. degrees from the University of Wisconsin, Madison. His e-mail is Stephen.Skedzielewski at intel.com.

Carole Dulong is a senior researcher and computer architect in the Microprocessor Technology Lab. She leads a team of researchers working on various data-mining techniques. She joined Intel in 1990. She was a member of the IPF architecture definition team and contributed to the IPF compiler design. She graduated from Institut Superieur d'Electronique de Paris (France). Her e-mail is Carole.Dulong at intel.com.

THIS PAGE INTENTIONALLY LEFT BLANK

Understanding the Platform Requirements of Emerging Enterprise Solutions

Krishnamurthy Srinivasan, Digital Enterprise Group, Intel Corporation
Raj Ramanujan, Digital Enterprise Group, Intel Corporation
Michael Amirfathi, Enterprise Platforms Group, Intel Corporation
Enrique Castro-Leon, Information Services and Technology Group, Intel Corporation

Index words: emerging solutions, platform pathfinding, platform architecture, usage models, deployment models, performance characteristics

ABSTRACT

Given the long lead time to put an entire platform solution together, enterprise platform architects must be able to predict the intersection of evolving usage models, deployment models, and platform technology trends in order to meet platform requirements several years into the future. Platform architects and technologists are generally very familiar and comfortable with predicting platform technology trends but the same is not true for deployment and usage models. Hence, we find that most platform architecture development is primarily incremental and evolutionary until one comes up against a wall of some sort along one or more of the vectors. Being able to articulate the intersection upfront has the advantage of influencing all three vectors, thus resulting in an optimum solution. Due to this potential for inter-dependency, the process for developing the intersection is inevitably iterative and complex. In this paper we concentrate primarily on the two vectors that are least understood by platform architects: usage and deployment models. We present a list of key solutions being adopted in different vertical industries based on an extensive interaction with industry leaders. We discuss the business usage model trends and the technology deployment model trends across the industries. We describe how the emerging models are different in their characteristics from those prevalent today, and using several real-world examples, explain the platform implications. Two key trends in the data centers of large enterprises are “scale-out” and grid computing. Scale-out allows application solutions to be deployed over a multiple independent set of resources that are networked together, while grid computing allows flexible and dynamic provisioning of these resources to scaled-out applications. Both these trends are driven by usage and deployment vectors focusing on lowering initial costs as well as improving utilization, scalability, and availability of data center resources. As enterprise compute and communication needs become increasingly complex, platform solutions from Intel have a crucial role to play in determining the optimum solution for these emerging models.

INTRODUCTION

As a general rule, it takes about four years to design, produce, validate, and take to market computer platforms with the microprocessor development taking the longest lead time. Hence, it is critical for platform architects to gain an understanding of the requirements of the solutions that need to be deployed that far into the future. In this paper we focus on the emerging enterprise IT solutions that are expected to have significant market adoption in 2009-2011 and discuss their expected performance characteristics.

There are two forces driving the rapid evolution of IT in the enterprise. The first is the challenge of continuous and rapid introduction of new or improved business processes to gain and retain an advantage in an extremely competitive market. This in turn translates into a need for accelerated deployment of new information systems capabilities (e.g., real-time decision support). The second driver, motivated by the prevailing trend of flat or dropping IT budgets, is to either cut or at least contain IT costs. Enterprise IT departments, faced with the challenge to provide new business capabilities at lower costs, are adopting various software and hardware technologies to develop, deploy, and maintain a larger portfolio of solutions at a reduced cost. Among these technologies are eXtensible Markup Language (XML), automated data center management, and server virtualization that uses an abstraction layer that decouples a consistent logical view of the server to the application from the actual physical resources that are utilized.

In this paper, we present the key new business solutions emerging in different vertical industries such as manufacturing and retail (e.g., intelligent inventory management in retail and collaborative product development in manufacturing) based on an extensive survey by Intel of key players in those industries. We identify key *usage model* categories that are common to these solutions (e.g., real-time supply chain management, collaboration, and image processing). We also briefly discuss some of the emerging *deployment models* such as XML and grids in the context of the usage models for which they are relevant. We discuss the significant new characteristics of these usage model categories. For example, real-time supply chain management often requires running of both transactional and decision support operations in the same environment, and synchronizing the databases underlying disparate business solutions more frequently. The key difference from the prevalent solutions of today is that all these operations will be running concurrently in order to enable a business to react very quickly to a rapidly changing environment.

These new characteristics of the emerging usage models are changing how enterprise IT departments do capacity planning. We present several specific examples based on real-world solutions adopted by leading enterprises in different industries.

KEY EMERGING ENTERPRISE SOLUTIONS

Physicist Niles Bohr said, "Prediction is very difficult, especially about the future!" This is particularly true about IT solutions where high expectations of new technology are often replaced by disillusionment and practical compromise. Significant time and costs are involved in optimizing large solutions and obtaining repeatable performance characteristics. The penalties of optimizing a general-purpose platform for the wrong requirements are even larger. Hence, we need to balance the prediction horizon with the accuracy to ensure that we look far enough ahead to accommodate the platform design lead times while being accurate enough with our predictions. To achieve such a balance, we focus on the solutions that have made it successfully past the phase where only the "innovators" with an extraordinary tolerance for costs and failure are interested in them; and are being adopted by the mainstream early adopters who pragmatically balance the technical risks with business benefits (Figure 1) [1]. For these solutions, mainstream adoption is expected in 2009-2011.

Intel's Solutions Marketing Group works very closely with the leaders in various vertical industries such as manufacturing and retail to understand the key business capabilities they are trying to enable. These leaders are the early adopters in their industries (Figure 1) whose lead would be followed by others in the next five to ten years. Intel plays the role of a "trusted advisor" in helping these businesses develop an IT roadmap to deploy these capabilities, and it also acts as a catalyst to drive the IT vendor ecosystem to remove any obstacles in the roadmap.

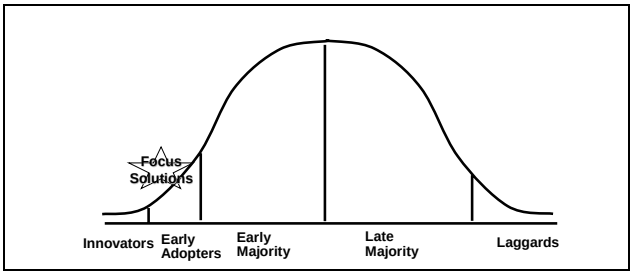


Figure 1: Phases of solution market adoption

Figure 2 shows the list of key solutions emerging in different industries Intel has identified. The list is based on the value of the functionality provided by these solutions to the businesses and their customers in the industry and the IT investment expected in deploying these solutions. While most solutions are industry specific, the requirements for mobility solutions are shared across industries.

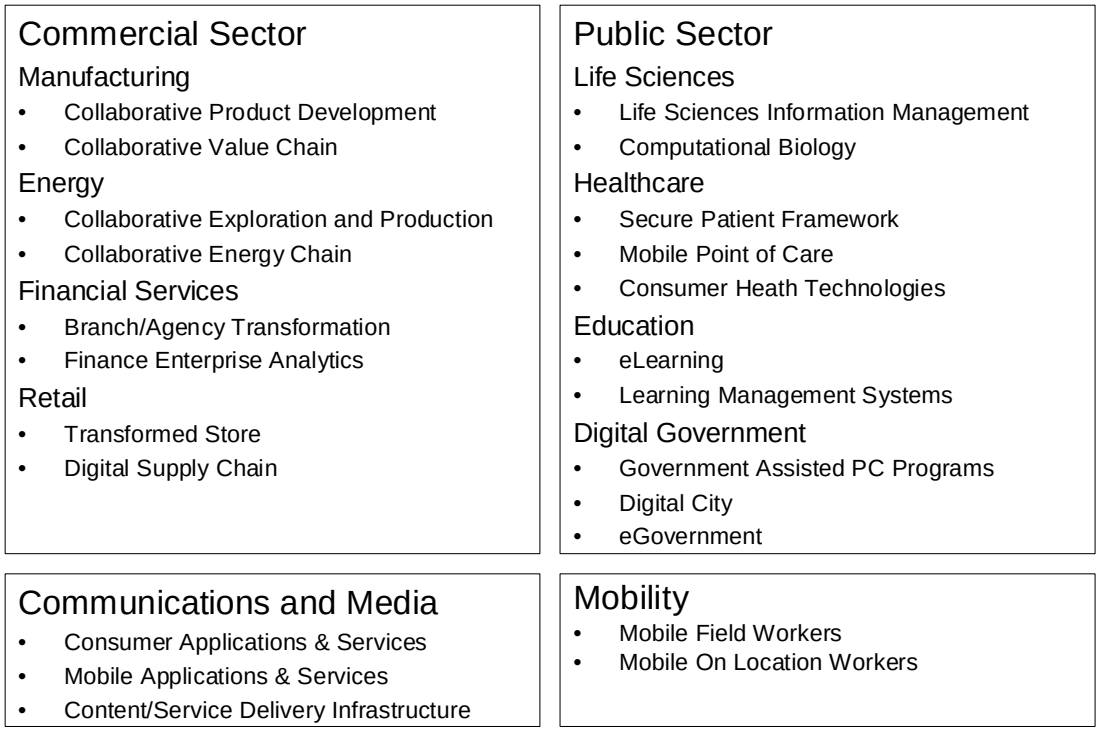


Figure 2: Key industry-specific and cross-industry solutions

USAGE MODEL CATEGORIES AND CHARACTERISTICS

Each solution shown in Figure 2 comprises two or more usage models. For example, consider the Digital Supply Chain solution in the retail industry (Figure 3). It consists of two usage models: the operational side of real-time inventory management and supply chain integration; and the analysis and decision-support side of making pricing and inventory decisions, and detecting trends and anomalies. Figure 3 shows two other solutions from the financial industry, their constituent usage models, and their categories.

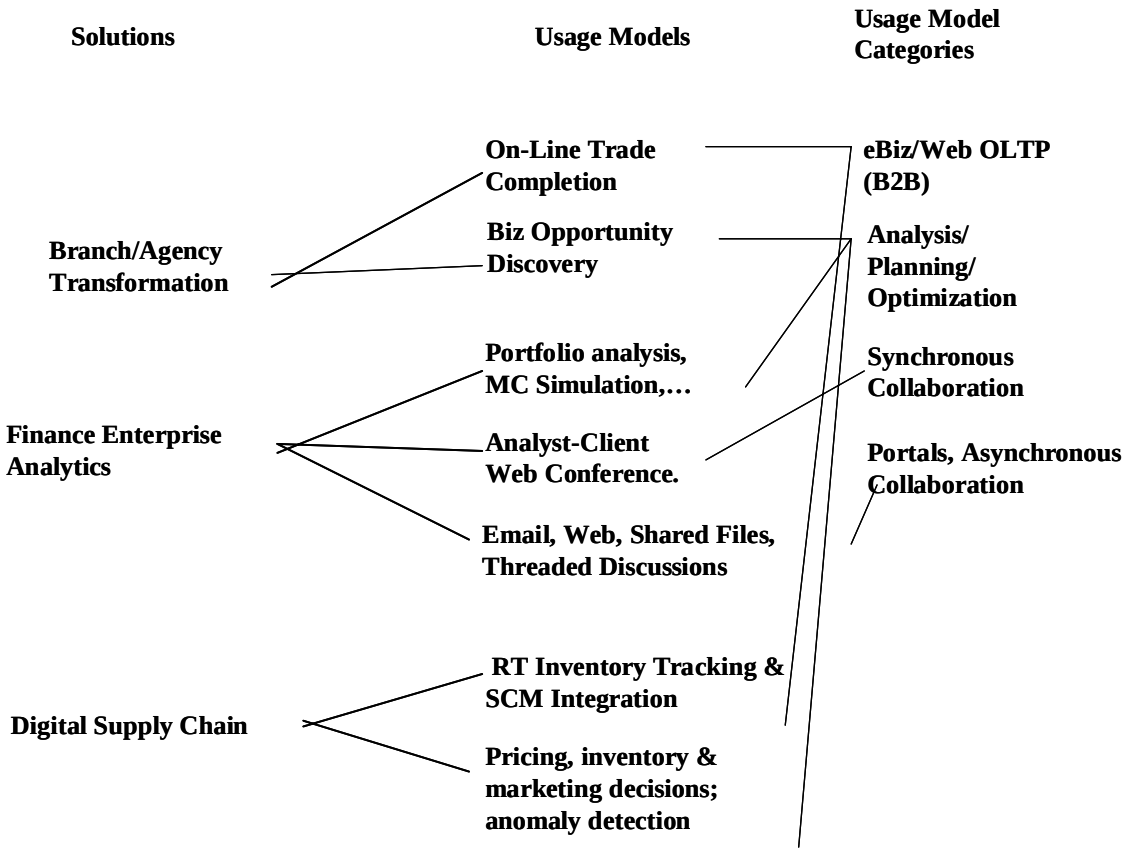


Figure 3: Identifying constituent usage models of solutions and categorizing them

Rationale for Usage Model Categorization

The usage models primarily help in identifying commonality across the solutions used in different industries. The solutions share a significant part of the software stack used to build similar usage models. For example, the eBiz/Web On-Line Transaction Processing (OLTP) usage model category invariably consists of Web servers, XML-based messaging, and links to enterprise applications and databases. Industry-specific applications (e.g., for on-line stock trading, student registration, or trading in electricity) might access these stacks in different patterns. These variations need to be comprehended in characterizing the usage model categories.

The second reason is the ability to map to industry-standard benchmarks from groups such as Transaction Processing Council (TPC) and Standard Performance Evaluation Corporation (SPEC). These benchmarks are widely used by IT hardware and software vendors today to evaluate the performance of the current enterprise platforms. Hence, it would be beneficial to use these benchmarks as the baseline and determine if and how the usage models differ in their characteristics.

Common Usage Model Categories

The most common usage model categories (based on how many solutions shown in Figure 2 they were part of) are shown in Figure 4. For each model category, we have also listed the forward-looking features that distinguish them from today's solutions. We have also selected specific solutions as representative of the usage model categories based on two criteria:

1. How aggressive a given industry is in pursuing the solution (e.g., the retail industry is a leader in adopting real-time inventory management while the financial services industry has been pushing the envelope in real-time analytics).
2. The IT market impact of the solution based on the IT spending expected for the solution.

The above information is based on an extensive survey of the industry leaders, conducted jointly by Intel and a research firm.

Usage Model Category	Vertical Industry	Solution	New Characteristics
eBiz/Web OLTP (B2B)	Retail	Digital Supply Chain	Real-time (Increased data volume, processing) Frequent DB syncs concurrent w/ OLTP Complex queries; app-level security, routing
	Manufacturing	Collaborative Value Chain	
Analysis	Financial Services	Channel Experience Consistency	Real-time DSS (Interactive, frequent/continuous ETL) Query across DB and XML Data mining Active data warehouse
		Wealth Management & Compliance	
	Government	Homeland Security	
	Manufacturing	Collaborative Value Chain	
Collaboration/ Portals	Government	e-Government Services, Digital City	Dynamic data gathering from DBs, Web services Rendering large content (>10s of MB) using XSLT Security to address privacy and trust concerns Voice interactions Federated ID to allow single sign-on
	Financial Services	Wealth Management	
	Manufacturing	Collaborative Value Chain	
Digital Content Processing & Delivery	Government	Homeland Security	Real-time/interactive image analysis Secure image storage & processing
	Health Care	Secure Patient Framework	
	Comm/Media	Content Delivery Infrastructure	
Technical Computing	Energy	Collaborative Exploration and Production	Real-time analysis Grids/Clusters Increased data volume/XML
	Manufacturing	Collaborative Product Development	
	Life Sciences	Computational Biology	

Figure 4: Common usage model categories

REAL-WORLD EXAMPLES

In this section, we describe real-world examples of solutions that embody some of the new characteristics. We also discuss their infrastructure needs and the software and hardware capabilities that would help in meeting the needs.

Real-Time Inventory Management in a Retail Chain

SAP described the increase in communication and computing needs of a retail enterprise with 1000 shops in migrating from a batch-oriented inventory management model using SAP's proprietary application messaging protocols to real-time inventory management using open, XML-based protocols [2].

Assuming a total of 33 million sales with three items per sale, the mySAP application communication from the 1000 retail shops to the central data center amounts to 2 GB per day, when the sales data are aggregated and communicated, once every 24 hours, using SAP's proprietary interface protocol.

Still aggregating and sending the data once every 24 hours, if the retail chain moves to an open XML-based interface, the data that need to be communicated and processed increases to approximately 20 GB.

Let us say the retail chain wants to react to sales trends faster. Aggregation delays the response. If information on every sale is sent to the data center to enable immediate response, about 200 GB of XML data has to be communicated and processed every day.

Here are some of the potential hardware and software capabilities apart from the increases in processing speed, cache, memory capacity and speed, and networking bandwidth that could help the retail chain implement the real-time inventory management capability:

- XML acceleration appliances, separate from the servers, may not help in all cases, particularly if large XML documents need to be moved. On-chip acceleration using special instructions or dedicated cores in a multi-core chip avoids communication latency overheads [3].
- Technologies such as InfiniBand Architecture (IBA) and Intel I/O Acceleration Technology could reduce the overhead in moving large amounts of data within the data center [4].
- Software changes to minimize or accelerate data type conversions or better leverage the hardware-specific features for handling different data types; and increased concurrency in XML processing could accelerate the solution.

Near-Real-Time Planning in Logistics

A logistics software vendor recently assessed the performance implications of supporting their customer's need to plan for filling orders on almost a continuous basis while simultaneously providing sub-second responses to the interactive RF terminals. This scenario exemplifies a couple of the new characteristics of the emerging eBiz/Web OLTP usage model:

1. Databases running more frequent synchronizations concurrently with the operational queries to ensure currency of data.
2. The same database also supporting complex, report-generation queries to support decision-making based on real-time data.

The minimum success criteria for this logistics software were 2.5 records/sec. for data download, 2.0 lines/sec. for order planning, and sub-second response time to the interactive users. Data download consisted of deleting a large number of rows in a database and inserting new ones. It was observed that a significant amount of time was being spent in deleting the rows. Using multiple threads for data deletion provided a significant reduction in time, particularly on Intel® Itanium®-based servers with EPIC architecture.

Currently, platforms are optimized for either throughput in transaction processing environments or for single job completion in analysis and planning environments. The industry benchmarks also emphasize one or the other. With the increased need to support analysis and planning in real- or near-real-time, such stove piping will not provide the best results.

Interactive Wealth Management Services

Traditional investment brokerage firms are faced with severe competition from Internet discount brokers who can charge lower transaction fees. The traditional firms are trying to leverage their strength in providing valuable financial advice to gain an advantage.

Charles Schwab, a pioneer in this field, wanted to provide a larger number of clients with objective financial advice at a fair price over multiple channels. They were faced with two requirements:

1. The wealth management solutions need to perform at interactive speeds for the financial advisors to work online with their clients.
2. The solutions need to be deployed on platforms based on standard, high-volume building blocks to keep the costs low.

A cluster of multiple IBM eServer xSeries 330* grid-enabled servers (using Intel® Xeon™ processors) using the Globus Toolkit for Linux*, reduced the processing time on the application of eight to ten minutes (and sometimes hours) to just 15 seconds.

® Itanium is a registered trademark of Intel Corporation or its subsidiaries in the United States and other countries.

* Other brands and names are the property of their respective owners.

® Intel and Xeon are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States and other countries.

The use of clusters of servers to solve complex problems, where processing capacity can grow incrementally with demand, is finding increasing application in various areas including CAD/CAE and chemical analysis and search. Such a scale-out model lends itself more easily to solving some problems more than others (refer to section entitled "Scale-Out Virtualization: HPC for Enterprises," below for details).

Secure Document Sharing for Global Manufacturing

A large manufacturer with factories and customers across the globe wanted to improve how it shared its product and process-related documents both internally and with its customers. It was faced with three challenges:

1. Delivering tailored documents to users instead of drowning them in documentation.
2. Minimizing the costs in generating and continually updating the documents with the rapid changes in products and processes.
3. Increasing concern over intellectual property and globalization demand a high level of security. However, poor usability and performance of the security solutions cause poor compliance by employees.

All these challenges were exacerbated by globalization. For example, this manufacturer dealt with a relatively small number of customers in North America and Europe. In Asia, they had to deal with a much larger number of smaller customers (more than 1000X).

To meet the first two requirements, the manufacturer moved towards a dynamic document-generation model. The data were retrieved from databases and Web services in XML format and rendered in HTML, PDF, or other formats for users to view.

Disk access was the first bottleneck in such a dynamic document delivery system. This was avoided by caching the XML content in memory. Rendering XML to HTML, etc., became the new bottleneck. These XML documents were typically in the 1 MB to 5 MB range. The eXtensible Style sheet Language Transformation (XSLT) scripts used to convert XML into HTML, etc., were typically in the 50 KB to 100 KB range.

The overheads in using specialized XSLT acceleration appliances prevented any improvements to them. However, scaling the solution out by adding Web servers to run the XSLT scripts was effective, as shown in Figure 5.

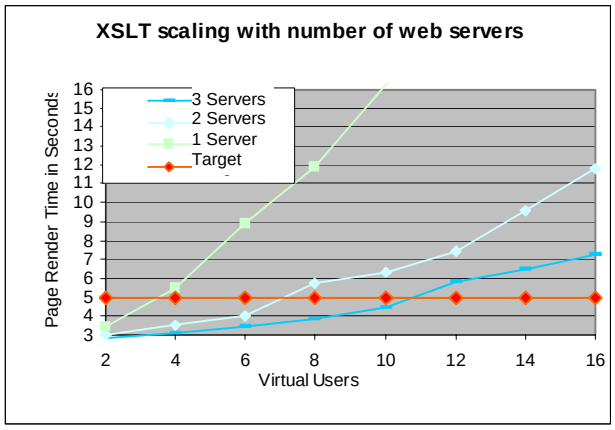


Figure 5: Response time improvement with the number of Web servers

To meet the needs of all its product groups and rapidly growing numbers of users across the world at a reasonable cost, significant speed-up in XSLT processing is needed. XSLT compilers and hardware support (without the overheads in moving the data around) have the potential to help.

The performance needs for security are equally significant. Currently, about 3000 documents are in a secure repository used by about 10,000 users. It is expected to grow to 100,000 documents used by 80,000 users in the near future. The personnel responsible for the solution think most of the users would use the secure repository only if the document access times are not excessive compared to non-secure access. Hence, they believe crypto acceleration is key to the success.

Other key requirements relate to ensuring the trustworthiness of the platform. They include securing private keys on both the servers and clients and ensuring that the client software stack is not compromised. The Trusted Platform Module [5] and La Grande technology [6] are expected to meet these requirements.

Scale-Out Virtualization: HPC for Enterprises

Grid computing technology has undergone a significant evolution over the past three or four years: the grid has been gradually moving from its High-Performance Computing (HPC) roots in university and government labs to more "mainstream" enterprise applications such as financial models and graphical rendering for motion pictures. We call grids applied in this domain "enterprise grids." Most of the enterprise grids are run on server clusters within the data center.

The concept of enterprise grids is literally server virtualization turned inside out, and it represents the next

step in workload disintermediation (decoupling applications from the physical platforms that run them): server virtualization allows multiple logical servers to run in one physical server. Each logical server runs one application. Conversely, in an enterprise grid environment, it is possible to apply more than one server, a *node* in grid parlance, to an application. We call this "Scale-Out Virtualization."

Enterprise grids of various sizes are getting deployed in different areas. Grids with 8-64 nodes are common for Computer Aided Design (CAD) and Electronic Design Automation (EDA) applications. Larger grids with up to 256 nodes are common in financial services, oil exploration, and pharmaceuticals.

Some of these problems are characterized as "embarrassingly parallel." It is possible to partition these problems so that computation and the associated data sets for parts of the problem could be isolated to individual nodes, and hence there is very little communication between the nodes. Monte Carlo simulation for investment portfolio analysis is an example of such a problem. Today's commercial servers and networks (100 MB or Gigabit Ethernet) could be used to solve these problems using large grids.

Other problems may not be so easily partitioned due to the need to move data between nodes or from memory to the CPU on a single node. EDA applications are examples of such problems. Messages could be "bundled" to minimize the penalty due to high network latency. This would require rewriting some of the applications. Alternatively, expensive interconnect technologies may be required.

Based on the data from several enterprise problems, we have derived a heuristic we call the Rule of 10: *the degradation in latency and bandwidth between two consecutive layers in the hierarchy should be no worse than a factor of 10 for all layers in a grid*. The on-CPU cache, main memory, disks, and network are the layers. For the "embarrassingly parallel" applications, this rule may not apply. For applications with significant data movement, the factor may have to be as low as 6. However, the Rule of 10 seems applicable to many cases.

Let us consider a cluster of servers connected by a Gigabit Ethernet as an example. The actual bandwidth for Gigabit Ethernet is about 100 MB/s. At the next lower level in the hierarchy, memory bandwidths of 3.2 to 6.4 GB/s are typical today in commodity servers. Hence, the bandwidth is degraded by a factor of 32 to 64. The degradation is even higher for latency: with memory latencies in the order of 100 to 150 nanoseconds, and the Ethernet network latencies around microseconds, the degradation factor is 700-1000. Hence, solutions with significant network communication need to be rewritten to bundle the messages, or the Ethernet has to be replaced by networking technologies with lower latency, such as IBA.

There are ways to work around the Rule of 10. Intel future platforms, for example, are expected to offer on-CPU caches comparable to today's memory in size, thus reducing the need for memory access and hence the importance of reducing the memory access latency for many solutions [3].

CONCLUSION

Wayne Gretzky, the legendary hockey player, once said that great hockey players go where the puck is going to be, not where it is. Well, the same applies to computing platforms. Great platforms meet what the solution requirements are going to be. Intel Corporation has identified the important solutions in vertical industries through extensive interactions with the industry leaders. These solutions were identified based on the significant value they offer to both the businesses that deploy them and their customers, as well as on the IT investments expected to deploy these solutions. We narrowed the solutions to those that are already being adopted by some leading businesses and are expected to be adopted by the majority of businesses in the 2009-2011 timeframe to balance the needs to be forward-looking yet accurate. We analyzed these solutions and identified the top usage model categories that are common to these solutions. These usage models differ significantly in their behaviors from the current solutions. Here are some examples of the differences:

- Competitive pressures to perform many operations in real-time (e.g., supply chain management, oil well analysis, medical imaging analysis, financial wealth management, and so on).
- Intellectual property, privacy, and regulatory compliance concerns, combined with globalization, drive a tremendous increase in the need for securing the platforms as well as the data that are being exchanged.
- XML, managed runtime environments, server virtualization, and autonomic management of the platforms are adopted increasingly to reduce the costs of developing and maintaining the solutions.

The above differences lead to new solution behaviors that are not adequately captured by the current industry-standard benchmarks. For example, the same database installation is increasingly required to support OLTP, decision support, and data synchronization concurrently.

We discussed several examples of real-world solutions that provide insights into these behaviors and their platform requirements. The current generation enterprise

platforms from Intel and its partners already meet some of these emerging requirements. Various technologies to support parallelism (e.g., Hyper Threading and EPIC), improved data center I/O technologies, large on-chip caches, and the technologies to improve the trust-worthiness of the platforms are examples. Intel's current directions towards multi-core chips, LaGrande technology, larger on-chip caches, and faster access to larger amounts of data in memory are aligned with the solution requirements trends we have discussed. These platform technologies, the increasing ability to manage the platforms and the solution stacks running on them, and scale-out virtualization will make enterprise grids a viable deployment model for a larger cross section of solutions. The future platforms could potentially offer even greater value to these solutions through features such as support for accelerating XML processing and cryptographic operations on the chip, native support for higher-bandwidth-lower-latency I/O, and higher platform trust.

ACKNOWLEDGMENTS

The authors acknowledge contributions from Anne Bartlett, Dave Dempsey, Heather Dixon, and Joel Munter, their colleagues at Intel Corporation. Thanks are also due to the valuable review and suggestions from Mark Chang, Jackson He, and Joel Munter.

REFERENCES

[1] *Crossing the Chasm: Marketing and Selling High-Tech Products to Mainstream Customers*, Geoffrey A. Moore, July 1999, Harper Collins Publishers, New York, NY.

[2] "SAP Position Paper," *W3C Workshop on Binary Interchange of XML Information Item Sets*, September 2003, Santa Clara, CA. http://www.w3.org/2003/08/binary-interchange-workshop/21a-W3CWorkshop-SAPPositionPaper.pdf*.

[3]Borkar, Dubey, et al., "Platform 2015: Intel Processor and Platform Evolution for the Next Decade, 2005." ftp://download.intel.com/technology/computing/archinnov/platform2015/download/Platform_2015.pdf.

[4]Grun, Paul, "The Changing Nature of Data Center I/O," 2003, http://www.intel.com/technology/pciexpress/devnet/docs/datacenterio.pdf.

[5] Meinschein, Robert, "Trusted Computing Group Helping Intel Secure the PC," *Technology@Intel Magazine*, January 2004. http://www.intel.com/technology/magazine/standards/st01041.pdf.

[6] "LaGrande Technology," http://www.intel.com/technology/security/.

AUTHORS' BIOGRAPHIES

Krishnamurthy Srinivasan is an architect in the Digital Enterprise Group in Intel Corporation. His current interests include understanding the platform requirements at various levels of the emerging enterprise solutions through performance characterization. Earlier, he played a key role in the evaluation and adoption of several software technologies in Intel's Planning & Logistics and Corporate IT groups. He made significant technical contributions to the development of Intel's Third-Generation e-Business vision. He managed Intel's Web service technology development and participated in industry-standards groups. During his sabbatical in 2002, he taught at the Indian Institute of Information Technology, Bangalore. He has a Ph.D. degree in Engineering from the Georgia Institute of Technology. His e-mail is Krishnamurthy.Srinivasan at intel.com.

Raj Ramanujan is a senior principal engineer in the Digital Enterprise Group in Intel Corporation and directs the platform initiatives and pathfinding activities. Raj is well recognized for his expertise and leadership in platform architecture definition. He has extensive experience in component (processor and chipset) micro-architecture complemented with experience in detailed performance analysis of architecture/micro-architecture alternatives. His e-mail is Raj.K.Ramanujan at intel.com..

Michael Amirfathi is a security architect in the Information Services and Technology Group in Intel Corporation. His current interests include understanding the enterprise security needs for protection of digital information and developing enterprise rights management solutions and services. He has extensive experience in architecting and developing enterprise applications for Aerospace, Finance, and High Technology industries. He has an M.S. degree in Engineering from Utah State University. His e-mail is Michael.Amirfathi at intel.com.

Enrique Castro-Leon is currently an enterprise architect and technology strategist for Intel Solution Services with 22 years at Intel working in OS design and architecture, software engineering, high-performance computing, platform definition, and business development. He has taught at the Oregon Graduate Institute, Portland State University, and he has authored over 30 papers. He holds Ph.D. and M.S. degrees in Electrical Engineering and Computer Science from Purdue University. He is also the founder of The Neighborhood Learning Center, a non-profit educational organization. His e-mail is Enrique.G.Castro-Leon at intel.com.

For further information visit:

developer.intel.com/technology/itj/index.htm