Intel Developer Forum
Morning Session, 9/28/06

STEVE PAWLOWSKI

[Beginning of recorded material]

[Video plays]

Male Voice: The history of supercomputing has always been that today's supercomputing problem is tomorrow's desktop problem. One of the things we're seeing right now is the introduction of a whole new capability in computational fluid dynamics that's called six degree of freedom problems. And what that allows us to do, simply, is to fly two vehicles in close proximity to each other and then to look at the interactions among the vehicles. And so, they don't just have to be vehicles, they can also be objects.

So whether it's debris coming off the shuttle that we want to track very closely and understand the dynamics of, or in the next generation of vehicles, we're hoping to supply true escape mechanisms that require extremely high fidelity in calculations using the six degree of freedom models. Those are exclusively supercomputing problems now and take the biggest systems we have. But we fully expect that in five years, those will be the kind of things that are being done by powerful desktop systems.

The current partnership with Intel has put NASA back on the supercomputing map. And that's exciting in allowing us to provide critical services to our engineering and science community. But even more exciting than that is a whole class of problems that we can't tackle yet, but we know we have to, in integrated vehicle design and sciences and in aeronautics.

Male Voice: Throughout history, industry leaders have always predicted that we've got all of the computing power we'll ever need. But history has shown that to be incorrect. End users are capable of absorbing all of the computing power that we're capable of delivering.

[Video ends]

Female Voice: Ladies and Gentlemen, please welcome Steve Pawlowski.

[Applause]

Steve Pawlowski: Thank you and welcome to the third day of IDF. We've saved the best for last. And I just told Richard that I'll start it off, and then it will all go downhill from there. Of which he promised, he said, "Hey, this is a subject near and dear to my heart. I know you're going to screw it up. I'll come out here and correct all your

mistakes." So, we're going to keep kind of a running score.

The subject we're going to discuss today is petascale, high-performance computing, but at the petascale level, and discuss some of the technological issues that we have to deal with as we start looking at petascale, and why it's important to us, why we think it's important. As you heard Walter Brooks from NASA, he's a wonderful guy, great speaker to come out and talk about what he does, what he's done with Columbia. It's the problems of today will be on the desktop of tomorrow. Now, not tomorrow tomorrow, as in tomorrow, but, you know, tomorrow in the future. And we'll discuss that somewhat.

So, before I get going, there are couple definitions I'm going to be using that just to make sure we're all grounded on the same page. And the first one is high-performance computing, or the acronym HPC. And it really has to deal with the hardware, the software, the tools, the languages, and the algorithms to process problems or to compute problems that heretofore have been uneconomical to do. And that's really associated with the fact that it's just been too -- the problems are too complicated. And we just haven't had the compute power to run those set of computations in an economical and efficient way. And those machines keep getting more and more capable. And we're able to add a new class of problems on those machines. However, it gets -- there's another class that will come in that just keeps asking for that next greatest machine.

The next one is petascale computing. And I shamelessly copied a definition that I think really resonated with me from Horst Simon on the 4th of August in a presentation that he did. And it really has to deal with not petaflop; it's when the widespread use of machines have a sustained performance of a petaflop per second or greater. So, the first petaflop machines will come out. They'll come out before the end of the decade. And that's great; they'll be muscle machines. But it's really when you get to the tens and hundreds of petaflops that the sustained performance is really at the level that you really need to compute these problems.

So high-performance computing. You're probably saying, "Yeah, those big systems. They do cost $50 million or a $100 million each." There are only three or four that get put into a data center. Why do you care?" Well, high-performance computing is actually tiered under various business segments. And if you add them all together, it's roughly a $10 billion industry, or at least there was $10 billion of revenue in 2005. And some segments are growing at greater than two digits, greater than 30 percent in terms of aggregate growth rate. So, it is a big issue. It's a huge market segment for us to be able to address. And it touches on the aspects of many of our lives.

Now, I always like this diagram because it kind of shows just how technology has progressed. 1946 with the ENIAC, a big tube machine. I think it took roughly a hydroelectric dam to power and cool; stored 20 numbers, but it was state of the art at the time. First supercomputer in the 1966, '65-'67 time frame, was from Control Data Corporation. And it roughly had a floating-point performance of about nine million operations per second. Now you fast-forward to 1977, ASCI Red was the first teraflop machine. And it was installed at Sandia Laboratories. It was based on the P6 at the time,

which was our Pentium II and Pentium III, early addition of Pentium II and Pentium III microarchitecture. Had 9,000 processors and supported up to a teraflop of peak performance.

Now move forward to today. We have a computing center in our Washington facility for evaluating benchmarks for high-performance computing [data] centers and applications. It's based on 400 Woodcrest processors, so it's roughly one-tenth the number of processors that the ASCII Red machine was, but it's 6.5 times the peak performance in terms of floating-point performance.

So as you can see, there's been a progression of technologies going forward and, as I will show in a moment, you will also see that we will get more and more of that capability sitting on our desktop. But before I do that, I'd like to bring Chuck Duvall out to actually show two systems that are moving to bringing high-performance computing capability to the desk side. Hello Chuck.

Chuck Duvall: Hey Steve. How's it going?

Steve Pawlowski: Terrific. So what are you going to show us today?

Chuck Duvall: Well, in the video we say earlier that today's supercomputers really are tomorrow's desktops. So I am indeed going to give you an example of two desktops, or really two machines that are supercomputers that sit beside the desktop. So if we come over here, I'd like to show you part of what's making that possible. And this is our new core architecture. This right here is based off Cloverton, so we have quad-core in this nice small package with extreme compute power and great I/O.

So we'll dig into the box a little bit here. We have a [TIAN] personal supercomputer that has 10 sockets. Those 10 sockets give me 40 cores. I tie those together with InfiniBand so I have great throughput that allows me to do some computing simulations that I've never really been able to do in a box this size.

I have an example; I'm running a mathematical that looks at pollution control. Really as a society, as we consume more energy, unfortunately we create more pollution. So we need to make sure that we can create the most energy, have the least amount of pollution. And the pollution that we do create, we find a way to minimize that impact on the environment. So we can run those types of simulations on this box.

Steve Pawlowski: So what you're really demonstrating here is you're running the same simulation, but as you're adding more and more cores to this box, the performance improvement is increasing substantially?

Chuck Duvall: Exactly

Steve Pawlowski: Okay. How about the other system?

Chuck Duvall: Now, this other system here, we have an SGI solution that looks at the other problem of high-performance computing. And that's memory throughput. You know, it's not just about cores and sockets; we have to be able to feed those. So in this box from SGI, we have an SMP memory architecture that takes all 12 sockets, which gives me 24 to 48 cores, and make sure that I have enough throughput to keep those processors fed. In this example, I have over 35 gigabits to feed those cores, which means those cores are running extremely efficiently and they're not sitting waiting on data. So I can run high-performance computing and not have to wait on the back end. Another great HPC solution.

Steve Pawlowski: Very nice. And so, again, you're showing that with adding cores as long as you have I/O scaling you get performance and then you'll reach a boundary at some point in time where you have to go through?

Chuck Duvall: Exactly.

Steve Pawlowski: What's this board here?

Chuck Duvall: Well, this is the board we announced yesterday. This is our Xeon 3000, our single-socket server. This allows us to increase density and lower power. So for volume computing, this is a great solution. We can put five of these into a 4U so when we go to quad-core, that'll be 40 cores into a 4U; great volume computing solution at a very low cost.

Steve Pawlowski: Very good. Very good. Okay. Thanks for the look Chuck, appreciate it.

Chuck Duvall: Not a problem Steve.

Steve Pawlowski: All right. Okay, so let's talk techie. If you go to the www.top500.org, you'll see this picture and it's a graphical representation of the top 500 supercomputers. The top line shows the sum of all top 500 in terms of peak floating point performance as measured by the Linpack benchmark. The bottom one shows the top machine. And if you draw a nice normalized line, it's been fairly linear in terms of the rate of performance on a yearly basis. So the Y-axis is logarithmic.

If you look at when we expect to see the first petaflop machine and you extrapolate out that curve, in the 2008-2009 timeframe, that's when we expect to see those systems available. Now, if you think in petascale, in terms of the definition that I gave, draw that line out a little more to when you get to 10, it's in the 2011-2012 timeframe. So a significant amount of performance is going to continue.

Now the bottom line is where the PC is, so what sits in your laptop today. And if you notice, there's about a 15-year lag between the performance of the top supercomputer at a particular point and when you see it in your standard desktop platform, single-core, dual-core or quad-core. So right now sitting here in the 2005-2006 timeframe, we're

roughly equivalent to what we saw in 1993, but significantly better than what you saw in the 1980s and '85s with the huge supercomputers from Cray, Control Data, among others.

So those problems, now, that we were dealing with at that point in time are also migrating and being executed on the platforms. So the transition from the technologies of the supercomputer, we generally see moving into the PC. And the capabilities in terms of bringing I/O bandwidth and whatnot, we have to bring those capabilities economically into the PC as well.

So the challenges. I'm a hardware guy. Richard will tell you that's a sin; don't listen to him. There are software implications here, but in the 28 minutes and 43 seconds I have left, I'll focus on some of the hardware issues that we're going to have to deal with and we can discuss software with David Cook and Richard potentially at another time.

So first and foremost, processor performance. Okay, if you draw the graph on our current processor road map of where we are today, and how floating-point performance has increased over time, it's also been fairly logarithmic, but it draws a nice, linear graph. And so, in the 2010 timeframe, when we get to a petascale machine, those machines will basically be, you know, if you follow the graph, in the 100- to 200-gigaflop range. The example I have here is 100. Well, if that's the case, then I want to build a petascale machine, a 10-petaflop machine, and I use those processors, I would need 100,000 of them. Now that number will keep coming back a little later, but that's a significantly large number, okay? So in order for us to be able to handle that, we need to increase not only the core density and get more cores to take advantage of a smaller footprint, but also increase the compute density per core.

So, we look at various options. We can take the current general-purpose cores that we have today and just try to put more and more of them on a chip. We can look at maybe getting more efficient cores that are built for that particular level of computation. We can mix and match them, because there are some things that we require scalar performance, a good scalar performance, like some of the collectives and barriers that are available in shared memory and distributed shared memory systems. Or you actually can use just a bunch of really special purpose, more highly optimized set of cores.

Now, if we map out the example that I have here -- and, oh, by the way, we also need to increase the floating-point performance of those cores. So, it's just not packing more of those cores in, it's actually making them more capable; a floating-point performance is really what you're looking for. So what's shown on this diagram is, if I have a standard die size that's 13 square millimeters, if I happen to take the performance in one particular vector, in this case scalar performance, and try to move that to be more efficient in vector performance, I will see a drop off in terms of what I can do on my integer workloads, or my scalar workloads, and that's what's seen on the lower hand, the bottom left.

However, if you look at the parallel performance, so what that really shows is each one of those cores, if the big core on the left-hand side and the 12-core configuration

is running a single thread in a small core, small core is roughly 30 percent, 40 percent as performance efficient as the larger core. However, if you go over to the right -- and you can paralyze these workloads significantly -- you see a tremendous improvement in terms of parallel performance.

When Justin, or Paul, showed the wafer of the Polaris chip, and Justin talked about it, that's essentially the direction they were going with that chip, which is lots of small cores, very simple, very capable floating-point units that you connect together. Now, we're going to have other issues that I'll talk about momentarily, but that's a direction that we think we can go. These are all the same die size. They're roughly 170, 180 square millimeter die size, but the capability you get out of those cores is much more significant if you're really focusing on parallel performance that we're looking at for the high performance computing applications.

So the next thing we've got to look at is how are we going to connect these cores inside? We got a lot of research going on in Justin's organization, and mine, to look at answering that very question. And there are different parameters and depending on the number of cores you have and the design targets that you're shooting for, the answer might be different.

If I were just to take these and say, "Performance is the absolute critical thing in getting bandwidth to the cores," forgetting everything else, I'd pick a cross bar. Now, a cross bar, essentially, is a mechanism that allows input agents and output agents to be fully connected without delay. A bus is, as it was described to me long ago, a bus is an idea of a poor man's cross bar, okay? There's a contention there. Cross bar has no contention. Very fast, very efficient, and for bandwidth scaling, as you add number of cores and increasing the size of the crossbar and the size of the switch, it actually is a good choice.

If I want to look at area, and the comparisons I have is a ring architecture where different agents sit on a ring and they transmit data in, you know, the ring could potentially be in two directions and they transmit data in both directions. Or a mesh, which is essentially I'm talking to my nearest neighbor, and they branch out and talk to their nearest neighbors and eventually we're all connected. Or the crossbar, like I discussed.

If I'm looking at just the area requirements, again, the crossbar looks really good, especially as I scale the number of cores. Where if you see the ring architecture, the area requirements for the ring go up as I scale the number of cores.

Now, if you look at power, different story. If you take into account the power, the energy required per ISO unit and bandwidth, that means equating bandwidth on every one of them, the crossbar wouldn't be my choice because of power. As I get up to even beyond the 16, but get up to the 32 and 64 core. So initial implementations which are in the 12-core range, may use something completely different than the 44, which may use something completely different than the 144, and it depends on the design targets that we

have.

So by increasing that processor performance as Justin showed, he said, "We've got a teraflop on die, having 80 cores on a die," okay? It's very good. Assume we get it to two teraflops in that time frame. We can put 160 cores. Now if I have a two-teraflop machine on per processor basis, I've gone down by a factor of 20. So now I can build a machine that's roughly in the size and footprint of the ASCI Red machine -- the Pentium Pro machine in 1977 -- but has four or five orders of magnitude greater performance with the technology of just being able to scale the number of cores.

But scaling the cores isn't the only issue; it's how do we get information into those cores. Justin, when he described Polaris, he also talked about stacking memory on top of the CPU. That's actually a good technology. You can get very high bandwidth because now you can be bump aligned with the processor and get very high throughput to the memory. But these machines require a significant amount of memory capacity, as well as bandwidth. Roughly, depending on the type of operation, I've heard half a byte per second per flop, or half a byte per flop on the average. I've also heard two bytes per flop. It just depends on the workload and how much of that data set that they want to bring into memory. We're not going to get that by stacking a single DRAM layer on top of the CPU. In 169 square millimeters, we're just not going to see anything like that.

So our issue is we have to maintain a high enough bandwidth footprint coming in from the memory subsystem inside this chip. Now we can do this by increasing the memory channel frequency. That has some limitations. We're pushing copper scaling to a limit, and then as you increase the frequency, the number of dims you can put on a channel obviously gets affected. We can put in a larger number of channels. That increases the size of the package, and it has a whole set of other issues that we have to deal with. Or we can try to come up with different and more exotic signaling technologies.

For Justin's teraflop chip, even though he's got a terabyte per second of memory bandwidth inside the chip, we roughly feel that we're going to need half that, or 500 gigabytes per second, off the chip to be able to handle the workloads that we anticipate these machines are going to need. That's a lot of bandwidth. There are different ways to get to it. There's a lot of research that's going on. But that's a challenge that we're going to have to at least solve from the memory point of view.

And as I mentioned, DRAM stacking is an option. It's a great technology, okay? They had SRAM on the design, but DRAM is certainly another option you can have. But do you use it as a cache, so you have a 128 or 256 MB cache? Or do you use it as part of the memory subsystem? And the answer is not obvious. Different workloads and different applications would require it to behave differently. And so those are the kinds of issues that need to be addressed as you start to partition the memory subsystem to be able to handle these new workloads.

The other thing is I/O. The general rule of thumb used to be if you had an

operation, whether it was a flop or mop or whatever it happened to be, for every operation you needed one byte per second per flop of memory -- flop per second of memory bandwidth, and roughly a tenth of that for I/O. Not necessarily the case. And I/O is becoming a significantly larger piece of the puzzle. And as we connect these larger systems, getting I/O and getting the latencies down for the traffic between different clusters and different nodes is becoming an issue as well.

Now yeah, we can scale PCI. We can add a full number of nodes. You can add infinite band and different types of interconnects. But again, we're talking about hundreds of gigabytes per second of bandwidth. When Justin talked about silicon photonics, it's a great technology, and we're doing it for a reason. We think there may be applicability in a space like this, and that's something that we'll try to address, okay? Yeah, it's good press, but at the end of the day, it's an excellent technology to start thinking about in terms of can it address the problems that we have from a bandwidth perspective?

All right, so we've got the memory, we've got the processor cores. Yeah, it's doable. There are some issues to be resolved there. We've got memory bandwidth and I/O bandwidth, and capacity that we have to deal with. Let's put that aside. Power and reliability. Data centers aren't going to get much bigger. If I had to build a 100,000-processor machine, just for the compute alone, we're probably looking at around 20 megawatts and that's not including the inefficiencies of getting that power in, stepping down the power, and also the power it takes to cool the facility and get that heat out. So, we really have to focus on, if we go build these machines, they have to be as power efficient or not consume any more power than the current processors we have today.

Now process technology will get us there in some respects, but in other respects, we have to be more clever in terms of how we do these designs and how we manage it. And this is just a quick example of, if I have to pay 10 cents a kilowatt hour -- I know at my home, I pay a little more, so I kind of wished I lived at this data center -- and the machine consumed nine megawatts, and I also include the power delivery, I'll never make that much money in a lifetime. That's $14.6 million. And if I have to build a bigger machine, at 100,000 processors, you could easily double that number.

And how we handle power has to really be handled at multiple levels. You've heard about strained silicon and how we try to essentially pull the atoms apart to increase the charge mobility and to also be able to reduce the amount of heat that's generated. We have leakage control techniques, there are [clock gating] techniques, and you all say, "Oh, great. But you're running these machines full out. Will these actually work?" Well, you know some pieces of the job finish sooner than others. That doesn't mean the machine has to sit full on all the time; you can potentially bring those down and control them at a finer granularity. There's we have policy-based processor allocation. Most of these machines run multiple jobs. In fact, it would really be interesting to have Walter come in and talk about what he does at Columbia and some of the simulations he runs, and the multiple jobs that he runs at any one particular time. So, there's policy-based allocation, as well as there are different things that we can also do in terms of how we would control power gating, not only at the processor, but at the memory and the I/O

subsystem level.

And then, again, looking at facilities, what we can do to become more efficient. DC power regulation was a concept that we talked about here. It's getting more and more prevalent. I'm getting a lot more interest from people coming in and saying, "Hey, tell us what we should be doing in terms of [bringing] DC to our rack and doing DC power distribution." There are some issues there that need to be resolved. Uninterruptible power supplies and battery backup are two key ones. But it's taking off, and if you can get the efficiency of at least just one conversion state, that's a significant power savings.

So reliability is kind of an interesting art. We talk about it; we use the term a lot. In a lot of cases, it gets misused. And particularly, here, reliability has to do with the number of times either a high-powered ion, a cosmic ray, alpha particles in a package, how many times that will cause a soft error, an error that isn't necessarily a persistent error, but causes an error which will affect the correctness of your machine and the correctness of your output. And that's usually measured in FITs, failures in times. So, one FIT is effectively one failure in one billion hours.

And so, as you see on this diagram here, as we pack more and more transistors in a given unit of area, the FIT rate per bit stays constant. So, if we double the number of transistors, the FIT rate doubles. Now, the nice thing about the concept of FIT is, you don't have to do all these exponential equations to come up with mean time between failures. You add them all up and then take the reciprocal, or divide it by one FIT times 10 to the ninth, and you can get your mean time between failure.

So, think of my 100,000 processor machine, and assume that I have a 400-FIT machine -- or 450, that was the number I calculated just before I came out here. 100,000 processors, that says I'll see an error in those processors -- that doesn't include the memory or I/O once a day. Most of these algorithms can't tolerate that. They'll do some work and they'll checkpoint, and they don't want to have to go back to the latest checkpoint and restart based on an error. And so reliability is something that is extremely important. And soft errors are something we have to focus on in our memory cells, not only on the chip, but also in the memory subsystem. And, effectively, transient errors that we will see across the interconnects, especially as we're pushing speeds in the hundreds of gigabytes per second across those links.

This is just an example of as the data centers grow, we are going to start dealing with this. I mean, even building a big supercomputer, just the mass of these mega-data centers that Justin discussed, those will have a significantly large number of processor systems. And those are systems that don't necessarily use these chips; they may just be our standard chips off the shelf. And we have to start building in the reliability structures that we put for these systems into those mainstream products. So again, technology for the high end that is effectively moving its way down into our mainstream product line, not something that we just keep as a boutique technology for the big capability machines that we put in the big facilities.

So what can we do? Well, we can go back to some of the standard techniques of doing very good protection across our memory structures, actually looking at potentially calculating parity as we go through the logic path and making sure that there's correctness there. There's a performance and a power issue associated with that, but you know, I think it's something we can at least deal with. Having ECC and better ECC codes, more robust ECC codes across all our memory structures, not only includes the caches, but the register files so that we can improve the reliability of the internal core.

At the macro solution, we may have to do things like redundant multithreading or lockstepping. This is basically dedicating a separate core to check and at the end of the computation, you look and see, did somebody have an error, and if they did, you take the computation of the other one.

Then, of course, at the process, we can always make parameter changes to the process. We may give up performance of the device in order to be able to improve a potential characteristic of that device, for example, our SRAM cells. Because if you were in the shop talk that we had yesterday, SRAMs tend to be a little more susceptible to soft errors than DRAM cells. And so how you build those SRAM structures and what those transistors look like, you may run it at a different voltage level than your processor; all those things need to be taken into account to keep those error reads fairly consistent.

Okay so you say, "Great. So you get to a petascale, why do you care?" Well, Stephen Chen, who designed the Cray X-MP, I believe -- still very active in the supercomputer business and the supercomputer field -- has basically mapped out where he thinks performance is going to go and where the performance requirements are going to be over the next several years, and the types of problems that we're looking at. As you can see, computational chemistry -- I couldn't tell you what it is, I can barely say it -- is going to take a zetaflop. That's 10 to the 21 flops, pretty significant amount of performance. And as we get into molecular dynamics, aerodynamic design, he has up there 10 to the 18. On and on and on.

So there's a significant amount of computation and computational ability that can bring to bear, that can actually improve how fast we can do design. It can help us in the medical field, in the biotech field, obviously in the aerospace. And we're only seeing this as being just the tip of the iceberg. But starting out and looking at the technologies we need to bring to petascale, we believe we can build on that going forward, still take advantage of the transistor densities we get through transistor scaling, which we believe will continue way into the next decade on silicon. And then be able to leverage some of the other technologies that we would look at for this type of system.

So I will leave you with our little diagram again, kind of extended out. And say, "Okay, when are we actually going to get this zetaflop machine?" If you map it out, the number one machine, we probably should see that in 2029. God, I'm going to be 70 years old. So I probably won't be working on that machine. But, you know, it's a natural progression. Effectively, every 11 or 12 years we see an order of magnitude of improvement in computing performance.

Now the interesting thing about this machine is we're talking about petaflop systems coming out in 2008. We expect in that same timeframe, roughly about maybe 2024, 2025, that your desktop will have that level of performance, if it continues the same trend that it has over the last several generations.

RICHARD WIRT

Female Voice: Ladies and gentlemen, please welcome Richard Wirt.

[Applause]

Richard Wirt: Good morning. Thank you. I want to talk something about software today. Steve set the pace for hardware and where we're going there. I want to bring you up-to-date with some of the software that we're working on. We're going to focus primarily on parallelization and threading. And most of this will be focused at the core level, as opposed to the big cluster level. A lot of the techniques are applicable across both the single CPU as well as the large cluster. Now in the cluster there are standards, open MPI and products like that, that we do support. So if you have questions at the end, we'll take those also.

At Intel, as you've seen, we're rapidly moving to more and more cores per die. We just introduced not only the dual processor Woodcrest, but we've now moved to Cloverton, which has quad-core. So automatically if you put two of those in a system with quad-core, you've got the possibility of eight threads that you want to use. The biggest problem for software is, how do you take advantage of these? And if you think about it, this is Intel's scaling strategy; to go more and more core. So if we don't get the software to go threaded and have a lot more of that available, ultimately the end user is not going to see a lot of benefit.

There is benefit in multiple processes running at the same time. I'm sure all of you now have your browser open, and simultaneously you may have your email. Those are separate processes; they both can share those cores and take advantage of them. And so there is a lot of benefit in those type of applications. Some of the applications that we're seeing, though, are very innovative and I think you're going to see a lot more of that as we move forward. So how can I simultaneously take conversation, translate it, maybe even put it in Word text, and data mine on that Word text as I'm running. Now this kind of application is easy to break up into multiple functions that each can run on a separate thread and, therefore, on a separate core.

The other thing that you do is you cut movies -- you watch a movie, you download it, you burn it to a CD. You get a lot of functional type things that you can split apart and make threads work here. One of the things that we did is we took advantage of virtualization and began to build a system. So we have just released, and it will ship out through some of the OEMs, software that virtualizes -- because of the virtual support in

the CPU, you virtualize the NIC driver. Therefore, you can begin putting manageability type functions on that virtual machine while you're running your other applications on the other virtual machine. So you have a way now of protecting the NIC driver and looking at all the data that's coming into your machine to look for viruses or management or whatever that.

So another functional splitting on the virtual machine, each virtual machine runs on its own core and therefore you can bring more value to that. Seeing a lot in the way of gains; really driving quickly to an immersion type of virtualization where you are immersed in the scene. You're also seeing that in Google Earth, Google Maps, you're beginning to fly around, go right down and a lot of innovative things happening here that you're bringing the power that you can do.

Give you a little update where we're at as work with ISVs and get the software threaded. Last year when we talked to you, roughly 60 percent level of the ISVs that we deal with were beginning to thread their applications. Doesn't say all their applications, that they now are trained or they've got major applications in progress; some of those are shipping, some of them are starting to. That number's come up, and you see it's come up significantly in some countries -- APAC running ahead of others. But now it's well over 80-85 percent of the ISVs are beginning to move their apps. So that's good.

Here is a list of some of those ISVs. You'll see on this list most major ISVs that you're building applications around now are in this threading exercise. Now, what we're doing with most of these is we have AEs that we've trained on our own processes and tools that we're implanting in these companies; training them, but also helping them as they begin to thread those applications or rework those applications. So that's progress that we've made.

In looking at scaling your performance, there are a number of things that you want to look at. And I want to just quickly highlight those and what they are. A lot of performance can be gotten out micro-architecture tuning. Now this is basically taking advantage of the micro-architecture, knowing information about cache size, all of those kind of things, the number of buffers to the bus and write-back buffers and all those things, and then being able to tune around those or utilizing them most effectively.

This level of optimization is typically the last thing you do to get the last 30 percent performance out of that application. This is also the part that we do well in tuning our compiler to automatically do this for you. That's the advantage of using some of the tools that Intel puts out. So whether it's our compiler or our hand-tuned libraries such as our math library or our video codecs, much of the effort we put in there is along these lines.

The other area is the application is self-tuning. Do I have the right algorithms? Do those algorithms effectively utilize the architecture? Do they effectively share data? How about bus bandwidth? Am I keeping the processor fed? So, typically, what you do here is you use a tool like VTune, you go through and analyze, you look at the critical path

through the application, where it's spending most of it's time, and you want to shorten that path. The shorter that path, the faster you get through that code and the faster the application runs. So that's a lot of what you're doing here.

In this case, we have provided the number of algorithms, particularly if you're into our math library algorithms -- a lot of uses of those in the talk that Steve gave. Most of those major players are using libraries like MKL; a lot of matrix solvers in that. An example of an interesting one that we've been working with is Johns Hopkins University. They're trying to do brain modeling to help a surgeon. As they do surgery on a brain, they want to be able to model the brain and get dynamic feedback of that. So, getting an application like that fast enough, that they can use it real-time is what it's all about. We're working with them on their equation solvers and trying to get highly tuned equation solvers that will speed that up. On an application like that, you can typically get 3x, 5x, maybe even 10x performance gain by choosing the right type of algorithm and then adapting your application to that algorithm.

Another type of tuning is system tuning. Now this is an area that's very important to look at, not only the network but the I/O sub-system. Can I get enough information in and out of that system to feed those CPU's? This is not between the memory and the main processor, but this is basically I/O in and out. A lot of work in the database area goes into this. You may be surprised to see that some of these benchmarks that we run, TPC at the very top, the high-end machines, we're running as many as 5,000 to 6,000 disk drives on those to get rid of the latency of the spinning disk and to get a CPU balance so you can really tune it. So these are the types of things you're worrying about there -- tuning that I/O subsystem and really getting the full performance out of it.

Today, I want to focus primarily on giving you an update of where we're at on our support for parallelization and some of the techniques we're looking at and working and actually delivering here. So if you think about it, this is where you take the application and you split it up so that it can take advantage of those multiple cores. So you should think of the core as the CPU's set of elements, registers, and that you run the applications on, ALU, and you should think of the thread as the part of the code that's running on a given core. So that terminology is a little bit loose, but that's conceptually what you have.

So what we want to do is speed up that application when it's running on those multiple cores. Now there are a couple of things that you need to look at here. First of all, we introduced hyper-threading, and hyper-threading basically was the first concept where you're actually duplicating a set of registers, but you're not duplicating the full chip. And you probably ask, "Well, why did we do that?" That's basically an easy way, an extra cost to do that, somewhere 5 to 10 percent in transistors, and is fairly low in power. So, that gave us -- keeps the power down, get performance, and we're seeing now, typically, we're after using that, somewhere in the range of 30 percent performance gain. It's the sharing of the cache, the sharing of some of the registers, ALU's, that limits that.

Now as we go to multiple core on a die, you've got the complete chip reduplicated, as opposed to just the register sets. This case you should expect the

parallelism to be basically somewhere, theoretically, 1x for each additional core. So two cores should be 2x. Practically, you don't get that kind of performance, because you still have the memory and you've got the bandwidth of the bus coming from memory.

When you think about threading, there's two ways of doing it. You can think of dividing up a task into subtasks and farming those out, or basically adding new functionality to the application. So, now if you're like me at home, your wife says, "You go do the dishes, and I'll mop the floors." That's two tasks, okay? They're independent. I can do mine when I get done watching TV; she can do hers before she goes to bed. One's not dependent on the other. So that's the way you want to think about it. Now if it's creating new capability, you're trying to bring innovation, then those tasks are different. It's "you go build a new addition to the house while I do the normal work." And you're bringing new capability and functionality there. So that's a way.

When you think about threading, though, really the problem is somewhat different. And the best way to look at is from a data viewpoint. I have this amount of data. I have to do work on that data. How do I divide my task to get that work done? We're going to show you some demos in a little bit. An example would be a video codec. Suppose I'm writing a video codec. I can break it up into blocks for each frame, and I can have a thread work on each block. Or, another way of dividing the data up is, I do frame one, you do frame two. I do frame three, you do frame four. You split it that way. So those are ways.

But when you think about it, you really want to think about taking the data, splitting it up. Now the problem when you do that is you can have data dependencies on the boundary. When I break it up into blocks, how about that little line that comes down the boundary? Who writes that? Do you write it, or do I write it? And there could be different results on that line. So how do you handle the data dependency areas, is where you have to spend a lot of time and thought. So that's one technique for threading. That's usually the way that you want to think about it if you're writing new code.

Another is functional. So this is like I mentioned where we split up the tasks into two independent things. And a lot of existing applications that you're doing, this approach to threading is where you end up, because you don't want to rewrite the full application.

So let me look a little bit at a couple examples. If I'm writing to the native threads of the operating system, or even a POSIX package of thread user level, there's a lot of replication I have to do. I have to start off by declaring the threads. I have to then create the threads. I have to then begin managing those threads and what they're going to do. So a lot of repetitious work. And the programmer, in his own head, has to keep track of this data does have -- or these two tasks have data dependencies and I've got to lock that data so that one guy doesn't change the value until it's correct, you know, if you're both trying to write at the same time, that's called a race condition.

Now there's one thing in this program I don't like. They hard-coded the number of threads in there. You see up at the top, constant integer, number of threads equals four.

We really encourage the ISVs, don't do that. You basically want to use CPU ID, ask how many threads are available in the system, and then use that many dynamically. That way, as you go to the next-generation processor that we bring out in six months or eight months, you automatically can take advantage of those threads, as opposed to being locked in.

So, that's traditional, a lot of handwork, a lot of keeping in your own head the dependencies and worrying about it. Why can't the compiler do this for you? It's meant to do the busywork. It's meant to check on those data dependencies. So, the holy grail of compilers for the last 25 years -- and Dave Cook is probably in the audience; he's the father of this -- was to produce automatic parallelization in the compilers. So what you want to think about here is [having] the compiler take that data, it figure out how to split it up, and then it looks for data dependencies, and it really then does all the work for you. That's done. There are compilers out there, including our own, that have auto-parallelization. Twenty-five years of research has improved it. You can get scalability that reasonable. Some applications have been rewritten to help that even do better. And we continue to invest in that area.

Now a lot of what we've done there, the infrastructure we've built, can be helped if the programmer gives some hints to the compiler. And that's what openMP is all about. It's an industry standard. It's in our compilers; it's in SGIs compilers; it's in Sun's compilers; it's in IBM's compilers; Microsoft's adding it to their compilers; it's in GCC. So it's an open standard that everybody is working together. And what are those hints? So those hints are those pragmas. Now, what it's telling the compiler is, "Here is the section of code that you can parallelize." And now what the compiler does is automatically figures out how to do a transformation on those loops, change the order around. So in this case you see, changes the order around so that you're operating on a column of that vector at a time. And it creates a thread for that column. So that's what OpenMP is all about. Let the compiler do as much of the busywork as possible, but the programmer give it some hints. This works very well for loopy code that has a lot of loops and that the data-dependencies, you can analyze it and figure it out. And if you know you're going to use this, you can write code better – even to make it better.

Now our compilers today hold the record. On SGI, I think it's 128-way SGI now, and we took SPEC – and there's a version the industry has of SPEC for OpenMP where it has these pragmas in it and how much parallelization are you getting out of it. So most of the scientific code, a lot of the multimedia code, most of our libraries we produce utilize this and try to get as much parallelism out of it as possible.

What I want to talk about next is where we're at on our tools and what's available. I talked to you last February and we had a number of things then in a beta program and now they've moved forward and they're actually moving into product. So the first new things are some basic building blocks. Now, this approach I'll go into some detail about, but at a high level, these are templates -- C++ templates -- and some existing algorithms that we've parallelized that allow you to write your code using our templates and that does a lot of the busy work for you.

The other the thread-checker, thread-profiler. These now are both up to version 3.0. We've gotten a lot of feedback from you on your applications and we've made improvements. Now one thing is, I pushed [on] my own compiler team, "Why don't you parallelize your compiler? Why don't you try to use your tools to parallelize the compiler?" "Bad idea. Why would I ever want to do that?" "No. Go do it." So they did. Guess what? It started off with somewhere in the range of 300,000 violations. Now, a violation is a data-violation; it's where you're trying to parallelize and you have those dependencies that you haven't fixed. It could be through shared variables; it could be through global variables. So they went through and began working on those. And now we have used this tool to get rid of all those violations. And we've got the basic part of the compiler now parallelized so that it will take large programs, break out functions, do those all in parallel, and begin getting some speedups by taking advantage of multiple cores at once to do that.

So we're trying to eat our own dog food here. We've got our AEs using them, feeding back improvements as they work with you. We've got our own teams trying to use them on threading some of our own applications; seeing some good results. Those are now both on Linux and Windows and some of these tools we are also moving to Apple. We'll talk about [that].

The third place -- we've done a lot of work on libraries. We can give you these heavy math libraries that are highly tuned, not only for the micro-architecture, but threaded; allows you to get your application to market sooner. We now are extending that to include XML libraries. These libraries are in progress. We're beginning beta. We're at the stage we would love your feedback. You've got a chance to influence us. We're trying to follow industry standards, [JAXBE], other standard Open Source libraries, so that you can just have a drop-in replacement for those interfaces.

We'll come back at the end and talk about those. So if you think about the entire tool set, here's the way we would like to think of the process to tune your application. First thing, go through and run VTune and the thread profiler and analyze your program and it's match to the architecture. Then you go through and there's those tools, and the version number tells you our latest shipping version, so if you're using them, you know whether you're on the latest version or not. Then you go through and introduce your threads.

Now when you introduce your threads, you probably aren't going to keep all those data dependencies in your head. So you're going to expect some programming errors. The next step then, comes along and helps you check those, the confidence, correctness, the thread checker. It helps you to see if you do have those data dependencies, that you need to put locks around, or if you have race conditions and you fix those. And then you come back after you've reduced all those problems, and you start working on optimizing and tuning, much like I talked about there, the system level, the architecture level, and the micro-architecture level.

So with that, that's the compilers and the tools that are available now. What I'd like to do is go into a little bit more depth on the thread building blocks. These are new. We've gotten some very good feedback on these, and we're looking at innovative ways to get more people using them, working together.

Now the concept here comes from a concept in C++. C++ handles what's called templates, and there are already some classes out there that take advantage and give you templates to build your applications. So things like caching, sorting, those kinds of things, are standard application templates that you can utilize. And those have been around as long as C++ and they're classes that go along with C++, and they've been standardized through the committees.

So what we said is, "Why don't we take the same concept, build templates, but go in and parallelize a lot of algorithms and provide lower-level capability to reduce your busywork." Why should a programmer have to initialize threads? Why should a programmer have to do a lot of management functions? Can we raise up to a higher-level capability much like OpenMP does, hence to the compiler, things like parallel for, reduce or scan, these kind of operations that you want to do in parallel. So, we've done that. [Parameter] is also for locks in that.

And then we took a number of standard algorithms and we parallelized them, so that we even provide that level. So let me give you an example here around the sort, and it's a quick sort. If you think about it, the concept I said, take your data and focus on parallelizing it and getting multiple tasks to work on that data. So the first thread in this goes through, picks a random spot, and does a quick sort on that random spot.

Now if you're not a programmer, it's pretty simple. I go through, I've got a random spot -- it was pretty close to middle; that's good enough. And that number turns out to be 37 [blade], yes. Now, once you have that, that first thread then, starts walking the list, serially. Is the number bigger than 37 or less than 37? If it's less than 37, it leaves it in the bucket it's in, your left side; if it's bigger than 37, it just throws it in the other bucket. So you've got an initial bucket of those two.

Now the interesting thing is, the algorithm divides the work and conquers. Now it has a second list it can sort, the yellow. So it spawns off another thread automatically and it says repeat the same process on that thread, or on that data. So, it goes through, finds a random spot and walks the list. If it's less than that number, it keeps on your left; if it's bigger, it throws it in the other side. And it just repeats this process down through, dividing the data and creating more threads.

Now, because we wrote the templates, we've put the busywork in to find out how many threads, how many cores are available and making it utilize all those cores that are available. So you don't have to do that; it does it for you and keeps track of that. The other thing is, we've built a scheduler in and basically, you have a pool of work to get done and you have cores that are available. So what the scheduler does is, it goes and grabs from the pool of cores, a core, assigns it to a thread, do some work. And you're

going to see that concept used quite frequently in threading. And that's all taken care of for you. So you're able to utilize this algorithm on your sort with different data, and it will effectively go through. Using these templates will do that for you. So, the process is on through until it's actually done.

So the whole concept of building blocks is built around this kind of concept. Provide templates for algorithms as well as some of the management-type functions and it allows you to go through fairly quickly. We've seen, in a lot of the applications that we've actually written, a reduction -- about a fourth as much code by the programmer needs to be written as if they do everything by hand by themselves. Also, the other advantage is the correctness when it's done. Somebody else is worried about that for you, as opposed to you having to go through.

So, I'd like to invite Charlotte Lawrence up to the stage and actually give you a visual demo of this on another system. Welcome, Charlotte.

Charlotte Lawrence: Good morning, Richard. I am so excited about our new product, Intel's Threading Building Blocks for C++. As Richard said earlier, it's a template-based set of libraries for multi-core performance and scalability development efforts.

So what we're going to do for this demonstration this morning is, I'm going to take the Tachyon application. And I have the native serial version. And, of course, I have a threaded version built using the -- what else? -- Intel Threading Building Blocks for C++. So, let me go ahead and get the serial version launched right now.

Richard Wirt: Can you tell us how many processors in each of these now you're going to use as you go through?

Charlotte Lawrence: Well, okay, the system that we're running this demonstration on is XeonMP dual-core platform. And this one has four processors in it, totaling eight cores.

Richard Wirt: Okay.

Charlotte Lawrence: So let me go ahead and launch the serial version, as I said earlier.

Richard Wirt: So being serial, this one's effectively only on one of those cores. So you can see the time on that.

Charlotte Lawrence: And, you know, when you look at this rendering, it's a very familiar rendering. Now let me launch the threaded version. Wow! Wasn't that just awesome?

Richard Wirt: Now I noticed there are a lot of black spots.

Charlotte Lawrence: Yeah. So that is actually what Richard spoke of earlier where you have the blocked rendering. And each of one those are the data blocks that Richard spoke of.

Richard Wirt: Effectively then, you've created blocks. And you noticed they were quite random in their pattern, so the library figured out, obviously, some stuff there, and assigned a thread to a block. And that block processed that data, and it did it in parallel, so you're breaking up a lot there.

Charlotte Lawrence: I had to just run it again. This stuff makes me really giddy. It is so neat. And, you know, Richard did say that what it does is it just goes ahead and does great performance. It scales well, but I think the really great news here is for the programmers. And, again, without going too much into detail, because, you know, Richard already covered it, is that you can actually deliver and achieve this application, and have the performance and scalability with much less code, definitely. We're looking at three-quarters less code using the Intel C++ Thread Builder than the native code threading.

Richard Wirt: Good. Understand you've got another demo. What's that?

Charlotte Lawrence: Yeah, you were speaking about the Intel C++ compiler. Let's get over here. Now we talked about this as a beta a year ago was it?

Richard Wirt: That's a silver notebook. I kind of remember most of our notebooks are black. What's that?

Charlotte Lawrence: Yeah, this is pretty hot. This is your Apple MacBook Pro, which is your Core Duo-based processor. What I'm going to show on this demonstration is, I'm going to use an Open Source, 3D ray-tracer application. And it's a binary that I built using the Intel C++ compiler. I also built a second binary using the GCC compiler. And I want all of you to know that I did use the best optimization that was available for both binaries. So let me go ahead and get this one launched.

Richard Wirt: Now, those of you who are not familiar with ray tracers, this is another way to give a 3D effect to a scene. And most 3D effects are through what we call rasterization, and it's going through and creating little triangles and coloring the triangles from the model. A ray tracer actually takes a ray of light and traces it through the scene. Now, that's a good example where, because it's a ray of light, you can have a single thread do a single ray. So it's very natural to parallelize that type of application.

Charlotte Lawrence: So as you view the rendering, you notice that the Intel binary, the binary that we built using the Intel compiler outperformed the GCC version, and we are looking at, I would say, easily a 30 percent performance advantage there. And I think that's just great.

Richard Wirt: So, thank you.

Charlotte Lawrence: All right. Thank you.

Richard Wirt: Thank you, Charlotte.

Charlotte Lawrence: So much.

[Applause]

Richard Wirt: This shows you some of the work that we've done to tune to that microarchitecture and really get the most performance out of that as we can. We also have people who are working on GCC and feeding back information there to keep it highly tuned.

So we've talked a lot about threading and how to go about it. The problem is most of you have existing applications and you want to take advantage of those cores. So one way to do this is through some of the third-party products that are available in the market. So I'd like to invite Geva Perry up -- I'm ahead of myself, just a moment.

One other library I did want to talk about is the XML, very quickly. So, much like the math libraries, we've threaded these libraries, and we're in the process of getting into final product. Now, in order to help you take advantage of these, we've tried to maintain as many of the industry-standard library interfaces for XML that are out there. So, Java AXP is a good example of an interface to Java for XML and allows you to process it.

Some of the results we're seeing -- these will be available over the course of the next six months to a year; they're in design now, as well as some of the early ones are in beta. Some of the results we're seeing on these benchmarks show you the 3x to 5x performance gain. Now, one of the things that we're after here is really beginning to establish these so we can put hardware under them. Most of you are aware we've bought a couple XML companies that are working in this space. This work is part of the Sarvega team that came on board. Also have a hardware team that we're beginning to really analyze how to speed up the hardware parsing, because a lot of data handling now is through XML.

So, I started to say, I wanted to welcome Geva Perry. Geva is an ISV from GigaSpaces. And basically they're focusing on existing applications, how you can take those and take advantage of the threads. Welcome Geva.

Geva Perry: Thank you, Richard. [Applause] Thank you. Thanks for the opportunity to let me speak here. We at GigaSpaces takes a somewhat different approach. It's a middleware approach that allows you, basically, to benefit from parallelization, whether within a multi-core, multi-CPU machine and across machines, without having to explicitly program for threading or for any aspect of the parallelization. We do it with an approach called space-based architecture.

So for those who aren't familiar with space-based architecture, I think I should start by defining what is the space in this context. That term applies to a distributed shared memory space. So I allocate a memory address space within a RAM of a machine, and I allow different processes and threads, whether they're local on the same box or remote, to share that memory space. So in other words, I can have one process, write an object into that memory space, another one read it, and essentially they share it. And I can cluster that memory space for high availability and for partitioning and load balancing.

Now how do I talk to this memory address space? And this is a key point. It's via standard API's, whether it's J2E type API's, like JDBC, so I can do SQL queries on this memory space, or JMS, or C++ objects, C-sharp objects, Java objects.

So, given that concept of a space, now what is a space-based architecture? So, this sort of gives you an example, you know, the orange rectangles at the bottom are basically boxes. A couple of them are multi-core boxes. One of them is a single-core. It's called multi-CPU, but the concept is the same. Essentially what I do is, I write objects as they come in and what are these objects? It's things that need to get done, whether it's data that needs to be processed, service requests that need to responded to, messages that need to be responded to, and so on. I write them into this clustered distributed memory space. And essentially it then, by itself, launches what we call worker threads. What is a worker thread? Well, a worker thread is your business logic, your code, that you wrote as if you're writing a single box, single CPU, single core, and it is launched as threads and can dynamically grow and shrink based on service-level agreements that you require. And those are the little orange rectangles at the bottom; those are the worker threads that basically work dynamically.

So just to give an example of what the effect of this is from a traditional approach to how you would build an application. In this case it's for a very large global bank, a foreign exchange trading application. So the basic logic of this application is, if I want to buy a million yen and sell for pounds, it has to match, find another trader that wants to do the opposite. In real life, it would probably be more complex, because it would have to find combinations in order to do those tradings. So it's a matching agent, for those familiar with financial services systems. Essentially it has to go through three things. It has to validate the order as it comes. Am I a legitimate customer? Is the order prepared okay, etcetera, etcetera. Then it has to do the actual matching logic. And then eventually you route it to different systems to execute the orders.

The way, traditionally, people on Wall Street and other places have built this is, as the orders come in from the clients, there is some kind of a validation logic that goes on in order to scale. What you would do is you would use a large, expensive SMP box and have to write proprietary code, etcetera. And then you probably want to write it into a database and that's how you maintain the state of the transaction. And then you would have another expensive box that needs to do the matching logic and work with the database. So there's a lot of latency there, inefficiency, complexity, and so on. And then eventually it will do the routing.

With this kind of parallel approach, I can do two things. One is I can have these three pieces of logic that are part of a sequence of a single transaction happen as thread and share that memory, share that data in memory so I get very fast performance. And it maintains sequence of the transaction. But what I can do now is I can parallelize that transaction, so I can have many of these transactions happening simultaneously because there's no dependency among them, as Richard was talking about before. There is no challenge there; each of them is independent. So I can do them across -- it views across cores and across boxes.

This is not science fiction. This is being used today in mission-critical applications in Wall Street and other financial services areas, in TelCos, in government agencies, and so on. It's real and it's happening now, and you can see some of the big customer names that I'm allowed to say who are using it.

So, basically to sum up, some of the key points about this is one, it allows you a way to parallelize your applications and run multiple threads and benefit from the advantages of a multi-core system without having to explicitly code for it. You get huge latency benefits because everything is talking to memory locally. It's very simple, I mean, you use standard APIs, either for your existing application or if you're building a new one. And it's a different, much more simple, elegant model.

Richard Wirt: Thank you, Geva. It looks like a very interesting software that you have there that takes advantage of existing applications today and helps them get parallel very quickly. Thank you.

Geva Perry: Thank you, Richard.

[Applause]

Richard Wirt: So in summary, I think you see that Intel is working with the industry. We've made good progress, as we move forward, at getting more applications threaded. We're providing a very rich ecosystem of tools, libraries around this, and we'll continue to do that.

Parallel, I wanted to give you a quick update. We're continuing to work with the universities -- we've got 50, 60 of them now we're working with worldwide -- to begin to impact their curriculum to teach more about threading and take advantage of multi-core architectures and this shift in the industry and trending. In parallel, we have our own training programs through what we call Software College. We put all our AEs through it and we're now in the process of going out and making that material available to universities as well as software parks, or within companies if you want training in-house.

I think you can see that ISVs and end users are now significantly engaged. A lot of new innovation is happening around this. We're very excited about the trend we're seeing in the industry and the capability that Intel is doing to continue Moore's law in

going to more cores per die and the fact that the ISVs are responding, bringing innovation to those, as well as scaling their applications.

BRENDAN TRAW

Female Voice: Ladies and gentlemen, please welcome Brendan Traw.

[Applause]

Brendan Traw: Good morning. I see that we have a fair number of hardcore attendees left for the final tech insight this morning. I want to spend a little bit of time sharing with you our digital home vision around Intel Viiv technology. And then I want to look forward at three key areas.

Unlike some other technology efforts, in the digital home group we're really starting with the consumer and trying to understand what the consumer wants in the way of technology in their home, as opposed to perhaps the more traditional technology-driven approach where you start out with some technologies and then try to figure out what they might do for consumer or other customers.

And so we've really been thinking about the digital home from a very customer-centric perspective. And one thing we know the consumers really enjoy doing, and that is being entertained. It's one thing to work or have personal productivity and the like. It's another thing to come home from a busy day, kick back with your family, and relax, and let the equipment in your home entertain you and satisfy you in those ways.

So, we focused on five experiences with Intel Viiv technology. Over on the right-hand side there, there are two experiences in the home, enjoying your entertainment content from your couch. So this is basically ensuring you have a set of user interfaces and the like to facilitate an experience where you're sitting on the couch, remote control in hand, watching your personal photos or perhaps a movie, or some other content. And this would be with a PC actually sitting in the living room next to the TV.

And then similarly, you can remote that experience. Your PC might actually be in your den because you perhaps do some personal productivity applications, your taxes, email, surf the Web, or whatever, in your den, but you still want to be able to take the content that you have stored on your PC and still enjoy it in other rooms of your house, such as your living room. And so the second experience there is streaming that content using that same 10-foot experience to the other rooms of the house where you wish to enjoy that content, not just where the PC is actually located.

There are some other experiences that we're interested in enabling as well that enable you to take your content with you. There's an experience we call synch-and-go. And basically that experience involves taking content that you already have on your Intel

Viiv technology PC and transferring it to another portable device. Perhaps this is a Centrino notebook. It might be a portable media player. It might be, you know, some other device that you have where you actually want to take that content experience with you.

You can also download content directly to a device, like a Centrino mobile laptop, and then take that content with you as well without actually having the Viiv technology PC, sort of in the path of delivering the content. And that's basically the fourth scenario there.

And the final one is perhaps, I don't know, infamously referred to as a sneaker net. And that is there may be places where you only have the ability to play back optical media. You may be going to a family member's house and want to be able to take a photo album with you so that you can all sit around and look at the photos from the trip you took together. And so you can actually take your photos or other content, burn it to a DVD, take that DVD with you. And then basically anywhere you have a DVD player, you can enjoy that content.

So to actually enable those user experiences, those compelling user experiences, a number of very important technologies need to be in place. The Core 2 Duo Processors offer a lot of advantages when it comes to actually delivering these experiences. You've probably heard a lot this week about the performance of Core 2 Duo in terms of both its raw processing capability as well as its power efficiency.

Another really important aspect of Core 2 Duo for the digital home is the fact that it's actually two processors. And if you've looked at those usage models that I just described a couple of minutes ago, you could readily imagine situations where you might want to be doing more than one of those things at a time. You might want to be watching a movie playing back locally on your PC while you're wanting to do a synch-and-go to your laptop, or to another device, so that you can take some content with you, perhaps content you have DVRed on your PC. You might want to take that content with you on your upcoming business trip. And so while you or your family are watching a movie in the foreground, you can use the other processor in a Core 2 Duo system to actually do the transcoding and prepare the content, the television content you recorded previously to actually take on the road with you.

Connectivity is also really critical enabler of these usage models. There are really sort of two forms of connectivity. There is connectivity to services. So this is actually enabling the flow of content from the Internet or from other sources onto the PC. And then there is also the need to connect the PC to other devices. Obviously, the streaming experiences that I talked about earlier, Sync and Go, these other experiences involve the interaction of multiple devices. And I suspect we all envision a future where your cell phone, your portal medic player, your notebook, your desktop system, your digital, all of these devices are sharing content back and forth. And that's clearly part of our vision with Intel Viiv technology.

Finally, it's important to add technology to actually make the consumer's experience better and simpler. In many cases people, you know, have this intuitive feel that if there's more technology involved, somehow it's going to be harder. And our philosophy with Intel Viiv technology is just the opposite. And that is, we want to add technology to actually make the experience better and simpler.

So whether it's insuring that the platform is capable of supporting high-definition content, both video as well as audio, or providing better ease of use by being able to instantaneously turn the system on and off, or the ability to use a remote control with the PC as opposed to have bring a mouse and a keyboard into your living room. Intel Viiv technology delivers a number of key technologies, again, to make that experience better and make it easier for the consumer.

So let's actually talk some numbers here. And again, you know, I had the generalities there on the first slide in terms of a 40 percent improvement in performance and a 40 percent improvement in power efficiency. Let's actually look at some of the workloads that are going to matter to consumers in the digital home.

So if you look at home videography, basically preparing your home videos to enjoy on your Viiv PC or, ultimately, to write out to a DVD. As you go from generation to generation here, you see very significant increases in performance. And, you know, in fact going from Pentium 4 to a Core 2 Duo, you see almost a doubling of performance, or halving the amount of elapsed time it takes to actually perform the operation. Similarly, for preparing your library of digital music, you see similar increases in performance, there again almost doubling -- or actually, more than doubling performance. Same thing with digital photos or transcoding content so that you can take it with you. Again, dramatic increases in performance across the board with Core 2 Duo. And if you then see how those map back onto the usages, you can see that this performance really does translate to an improved user experience. It's not just numbers; it's really directly translatable into what the consumer is actually going to see.

So obviously, having a compelling product is more than just having a powerful processor. You also need to have stylish solutions. People are going to be including these products, actually, out in the living space of their home -- in their living room and in other visible places in the home. People are going to want something better than just a beige box.

And you've probably all heard Paul Otellini introduce the million-dollar Intel Core challenge earlier this week. I'd like just to quickly refresh everyone's memory about this and hopefully stimulate some of you to actually go and enter into this contest. And so what we're actually looking for here are innovative form factors for Intel Viiv technology PCs using Core 2 Duo processors. We are looking for form factors that are smaller, quieter, and more stylish than what is done today. And we hope we can get a number of submissions from ODMs and OEMs across the industry and from around the world. And we'd like those entries to be submitted by January 15th, and then there'll be a judging panel of distinguished experts, not only from the technology space but also from the sort

of the style space, who will be looking at those submissions and will be making a decision in Q1 as to who the winners are. And there are some fairly significant financial rewards for the couple of companies that win this contest.

So again, I hope everyone can put their thinking caps on, and let's really come up with some new form factors and some new ways of packaging PC technology so that people are -- consumers are going to feel proud to have this everywhere in their home. Not just in the office, not just in the kids' bedroom, but really everywhere in the home.

With that, let me actually move now to giving you some insights into three critical areas of digital home infrastructure that we're going to need to deliver over the next several years in order to continue enhancing the consumer experience around digital entertainment.

The first is with content delivery. And, again, I mentioned earlier that Intel Viiv technology is about providing connectivity with services and with devices. And the services that I think immediately come to people's mind are content-delivery services. And I'll talk about a second type of service later in this presentation around manageability, but let me start with content here.

And specifically, you know, one of the great, exciting things about PC technology and the Internet is the opportunity to gain access to the entire world of content. It's about connecting content creators with the consumers of content, the customers around the world. And, you know, as we've all probably found with Internet content to-date, there's a little bit of something out there for everybody. And you can actually target content at a fairly small niche of people. And, you know, this is something that your traditional cable operator or satellite broadcast operator or even over-the-air broadcast operator really can't deliver, they can't deliver that customized content for each and every person that subscribes or receives their service. However, with the Internet, you can actually create those connections between individual content creators and individual content consumers because there is no sort of incremental cost associated with actually providing that additional connectivity.

However, for a content provider to actually get their content on the Internet and be able to deliver it to consumers, and particularly be able to do that in a way that works well with a 10-foot user interface and able to take advantage of the remote control and all of the other aspects of Intel Viiv technology that really make that a great consumer experience, that actually takes some work on the content owner's part. And some of the initial folks that we've worked with have had to spend several million dollars, in some cases, actually building the infrastructure and the user interfaces and the other pieces necessary to actually get the content, whether it's sports, movies, music, television, games, whatever, to the end user. And there's a number of different pieces that have to be in place in order to get this, and one of the things that we've been working with the ecosystem on is actually putting together turnkey solutions, so instead of basically reengineering the solution each time for a given content owner, content owners should be able to go to basically a middleware or service providers who can then help facilitate

them getting their content onto the Internet in as efficient and time-effective way possible.

So let me move forward here and actually overlay some of the specific pieces of infrastructure that are needed here. And, again, you need to be able to take the content in its original source form, you need to be able to encode it and encrypt it so you can apply DRM and attach the business rules to it. You need to be able to add metadata to the content. Metadata is absolutely essential, because that's actually how consumers will be able to find the content by searching on the metadata using a search engine or other capability to actually identify the content that they want. And, you know, one of the things I think people undervalue is this metadata, and from my perspective, it's really sort of garbage in, garbage out. If you don't have good metadata associated with the content, then you're going to have a very difficult time actually having the consumer figure out which content they want. And if the consumer doesn't have a satisfying search experience, then they're unlikely to actually continue the experience. So I very much encourage people to really put a lot of effort into the metadata. It's, in many ways, from my perspective, as important as the content itself, because without good metadata, you're never going to hook up the right consumers with a given piece of content.

So once you've got all that done, you need to then be able to actually get the content hosted on the Internet and be able then to push that out to the consumers. There's, of course, a number of other ancillary capabilities that are needed, depending again, on the specific business model and the specific type of content you're distributing. This includes everything from financial transaction clearing systems to actually take the payments, you know, some content, however, consumers won't pay for, and instead is perhaps supported with advertisement and other things, so you need technology to insert advertisements and basically to fulfill the business models. And so basically you end up with a situation where there are quite a range of components that are needed, and content owners, in conjunction with their service provider, need to be able to draw on these turnkey solutions in order to, again, reach the market in a very effective and efficient way.

So with that, I'd actually like to turn and do the first demo here, which is going to be showing a new content service that was actually just announced today, and if we can run the video here. This is from NBC Universal, it was just announced this morning. This is part of the Viiv Entertainment Pass. And basically what they've made available here is a very compelling set of television content, including TV shows that they're showing this fall, and in many cases, letting people preview those television shows.

One of the exciting things about this is this is actually delivered for free to the consumer. It's ad supported here, and so there's actually a pre-roll where a couple of advertisements play, and then it moves directly into the television show. Actually, there's a preview here as well, but then it will move to the television show. This is actually live on the Internet today. If you have a Viiv technology PC at home, you can go home after IDF and enjoy this content just as we're doing here.

One of the other things that really excites me about this is the quality of the visual experience. This isn't your typical, you know, sort of sub-VHS VCR quality content. This is very high-quality video that looks good even on a large screen, even a screen that's many feet across.

So with that, let me -- actually, kill the video, and let me move on here to the next of the three areas that I would like to focus on. And that is home storage. And I suspect that everyone in the audience here is in the process of taking basically their entire lives digital. And to the extent you're taking your life digital, you're entrusting basically very important elements of your life to the digital technology that you carry with you, you have in your office, and you have in your home. And I'd like today to spend a little bit of time talking about some of the challenges with the current digital storage environment in the home and some approaches for dealing with these problems going forward.

The first problem that I'd like to identify is the fact that there are many disjointed islands of data around the home. I look at my house, and I have seven different hard drives that are in use on a daily basis. A couple PCs, a DVR, an iPod. I suspect actually many of you probably even have more than seven in your home. And right now I'm very hard pressed to be able to basically enjoy the data or the content that's on any one of those arbitrary storage places anywhere else in the home because basically all of the devices are not connected together, they're not interoperable, and you're not able to basically get access to the data you want when you want it.

The scary thing is, is as bad as it is today, it's only going to get worse here. The number of devices is going to continue to increase. Again, if I look at my own personal home as an example, the number of hard drives I have has basically doubled in the last 18 months. I know that's starting to sound like a Moore's-Law-style trajectory, and I think that's a pretty scary proposition. Hopefully I don't have another doubling in another 18 months, but nevertheless, I would suspect that by the time I come back to IDF next year, I'll have 10 or 12, you know, hard drives, major storage places in my home. I suspect, again, all of you are seeing those trends, and we're certainly seeing it from the analyst data, you know, looking at the deployment of digital technologies around the world.

The second part of this that's a really significant challenge is the actual number of data objects that you're trying to store in your home is dramatically increasing. You know, as you go from, for instance, film cameras to digital cameras, the number of pictures you may take in a year may be thousands of pictures. Being able to find the ones you want can be a real challenge. You add that with all the content you're DVRing, your home movies, all the other digital stuff in your life that's important to you and you can very rapidly have just a phenomenal amount of data, not just in terms of the number of bytes of data, which I think is going to rapidly approach the multiple terabytes for most consumers, but also the sheer number of actual individual data files that you need to be able to somehow navigate through to get to what you want. And, again, that very much goes back to my earlier comments about metadata and the importance of having good metadata. And I know that's been a personal failing of mine with digital photos is that I don't go and put the metadata in with the photo, and instead, you know, basically have to

rely on the date stamp or whatever metadata the camera inserts automatically for you. Wouldn't it be great if you were actually able to, if you wanted to see pictures of your children with their grandparents, be able to just enter that in and get the photos you're looking for, as opposed to trying to figure out while, gee, my grandparents were last visiting on the third of August, and I've got to go back there and try to poke around and find the photos I'm looking for that way.

Basically, the bottom line here, you know, users aren't going to be able to find what they want going forward here. It's going to be looking for a needle in basically a number of very, very large haystacks. And so I think there are some really very fundamental things that we need to be doing in this phase around metadata creation, around the ability to search metadata, and again, making sure that all of these disparate islands of storage in the home are connected and accessible.

Another piece of this is the permanence of the data. And if you look at the diagram on the left, if you look at the different types of contact you have in the home, they vary quite a bit in terms of how important they are to you and how hard they are to replace. And if you look at, for instance, your personal photos, in the upper corner there, this is data that is basically irreplaceable. If you lose it, it's gone. You can't get it back, no matter how much money you want to pay. And, you know, its data that you're going to want to last for your lifetime, or maybe even multiple lifetimes as you inherit, or you pass down your digital photos from generation to generation. That is an incredibly profound sort of thought that you actually have this data that is so important to you and this is something that you may want to have persist for, for decades or even longer as physical photographs have done.

Obviously not all content rises to sort of that level of importance to consumers. And if you look at commercial content you've downloaded, you know, for instance the content you've downloaded off of iTunes or whatever you've done, I mean, this is content that you could replace. It would be pretty annoying to have to replace it if you lost it. It might be expensive to replace, but nevertheless, you could do it. All the way down to sort of temporary files and all, where they're really only important for sort of right now and if you had to replace them or sort of roll back to a previous version, it wouldn't probably be the end of the world. So again, though, you have this incredible range of, sort of the importance and the difficulty of replacing content. And we really need to, with the home storage, be able to address that full range.

Now, of course, the primary device that people store content on in their homes is a hard drive. And if you look at the chart on the right there, this is looking at the cumulative probability of having a hard drive in your home fail in any given year by the number of hard drives you have. And as you can see here, as your number of hard drives increases, your probability of having a hard drive in your home fail in a given year increases pretty dramatically. And so, in my case, I have about seven hard drives in my home, and that's, what, a 35 percent chance in any given year of having one of those hard drives fail. As people move forward here and have more and more, it's going to be more and more likely to have those fail. And unless these devices are connected and mutually

backed up and have the ability to rely on each other to actually deliver the permanence for your data, I think consumers are going to start to have some backlash here. I know personally, if I were to lose our family's digital photos because the hard drive they were stored on was to become corrupted, this could be a very significant event in the relationship between myself and my wife. And I suspect a number of you would be in a similar state if those important items were to be lost.

The choices that consumers have today frankly aren't very satisfying. Again, if you have a terabyte of storage in your home, which I suspect actually many of you probably do, you're talking about hundreds of DVD's to back that up and thousands of CD's. This isn't a very feasible thing to do. Yes, you may back up a subset of your data onto optical media. I know that's personally what I do. But I think what consumers are really looking for is the ability to actually have the home infrastructure step in on their behalf and actually deliver transparent solutions to protect that data, because, again, there are many hard drives in the home, many different devices. Wouldn't it be great if they could work together to actually deliver the sort of data permanence that consumers are going to require?

So, I hope I've made a case here for the ecosystem to work together with Intel, and, obviously, with the other key players to actually deliver a comprehensive solution for the home. There's some great work already started in the standard space with the digital living network alliance and the guidelines that are being created there in terms of file formats and interconnectivity standards between devices. That's obviously great infrastructure, a great place to start. But we need to do more. And we need to layer additional capabilities on top of it, like the mutual backup capabilities that I was describing earlier.

Additionally, there are some other opportunities here to improve the home storage situation. And you've probably heard about Robson Technology from the mobility keynote earlier this week where you add non-volatile memory to the platform. There's a number of exciting opportunities in the home to accelerate performance. During the spring IDF we showed, during the digital home keynote, a really exciting gaming demo that took advantage of those sorts of capabilities. And so I think we can not only improve the connectivity and accessibility of content and your important digital stuff by looking at the home storage from a broader perspective, but also improve performance and responsiveness of the systems as well.

Beyond the home itself, obviously the home is not an island in and of itself. Going out into the broader world, it would be great to be able to take advantage of resources that are out in the network, whether it's to provide backup capabilities for your essential digital stuff, or to enable you to access your content when you're away from home. When you're on a trip or visiting family, to be able to get back into your digital storage in your home and be able to access those items without having to explicitly take that content or take that digital material with you whenever you pack up for the trip.

So with that, let me turn to the final of three focus areas here, which is remote

management. One of the things that I think many of us benefit from, just as sort of tech-savvy individuals, is the ability to solve many of our own technical problems in the home. Unfortunately, the vast majority of consumers don't have that expertise. They're not able to, you know, look at the, you know, they don't necessarily have a technology industry family member who can sort of step in and be their home system administrator to fix all their problems. And one of the things we're looking at is actually developing technology to enable service providers and others to add value to the consumer's experience by actually stepping in and taking over that responsibility in the home, and helping people manage their increasingly complicated information infrastructure.

And, you know, this has some additional benefits, as well, for OEMs and retailers in terms of reducing the number of returns on products. I mean, it's not only frustrating to the consumer to bring a product home and not be able to make it work or have it do what the consumer thought they could have it do, but it's also bad business for the retailers and the manufacturers because that product comes back. And, you know, you've got to do something with it; you've got to refurbish it, or whatever, to actually get it back out. And in many cases, the product wasn't even broken. It maybe just hadn't been configured correctly, or the user wasn't able to actually get it working. And so, wouldn't it be great if you could come in down the wire and help the consumer with the installation of that product once they obviously got past the sort of fundamental piece of providing some connectivity to it.

Similarly, for the service providers, you know the bane of the cable industry is doing a truck roll, or the TelCo industry is doing a truck roll to install new capabilities or to fix a problem within the home. Again, wouldn't it be great if you could come in over the wire and fix a lot of those problems and not actually have to, you know, have the consumer be inconvenienced by having to meet a repair person. And then furthermore, the obvious expense of having to actually send someone out onsite to do the repair.

So, not surprisingly, there are some really good technologies to bring to bear in this space. And this is a place where, from a digital home perspective, we've looked at our colleagues in the enterprise space to look at the sorts of technology solutions that are being delivered for the enterprise that may be applicable in the home. And Intel's active management technology, or AMT, provides actually a really exciting baseline for delivering a home manageability solution. And the really nice thing about AMT is it provides and out-of-band path for actually interacting with this manageability engine. So, even if your OS can't boot because something's been corrupted or you've got a configuration problem, the manageability engine can actually create a connection on the Internet, on the network, and actually communicate with that external service provider and provide a view into the home PC to actually remedy these sorts of problems.

And I'd actually now like to turn to a demo showing a couple of capabilities in this space. I'd like to welcome Don to come out here and join us for a demonstration.

Don Bowden: Yup, I'll do that for you Brendan.

Brendan Traw: Thank you. One of the things I sort of like to point to is our progress from sort of PowerPoint to actually product. Last fall at IDF, I joined Don Bowden, who was the general manager of Digital Home Group during his keynote and talked conceptually about this sort of capability. And we showed a little bit of a smoke and mirrors demo of how it could look. Today I'm actually really excited to be able to show a product that we're developing with a Chinese ISV. This is [Star Soft Com]. And Don, can you sort of walk us through the capabilities?

Don Bowden: Sure. Let's start off by talking about our home system here. Now what we've done on our home system is we've developed some policies to monitor some of the activities on the system -- things like our firewall, our antivirus software, and what we're going to focus on particularly today is an application called NetNanny. Now, what we've done is we've gone into NetNanny, and with, you know, keeping our kids safe in mind, I've gone in and blocked a lot of the undesirable sites that are out there on the Web. So that's what we've done here. But my kids are pretty inventive, and so my son is thinking to himself -- and believe me, I know how he thinks -- he's going, you know, if I go in, I could just stop the NetNanny service and I'm going to be able to go anywhere I want.

Brendan Traw: Pretty clever.

Don Bowden: He's clever. He really is. So he goes in, and he's stopping the NetNanny service. I want you to pay attention to the lower corner down here to see what happens.

Brendan Traw: And I bet your son's told all his friends how to do this too, so even if they're not as tech savvy, it spreads like wildfire.

Don Bowden: Absolutely. So what's it's done is it's basically disconnected the system from the network. Once he stopped NetNanny, the policy goes on and says, wait a minute, NetNanny isn't running, I'm disconnecting the computer all together, so it's isolated now. So my kid's not going anywhere until he starts NetNanny back up again. So we'll go ahead and do that. And now once NetNanny starts back up, he'll be able to go out and surf the sites that I'm going to allow him to go to.

Brendan Traw: And, again, that's implemented using the manageability engine and AMT to actually monitor the system to make sure the necessary --

Don Bowden: Absolutely, and it works along the same ways with the firewall and antivirus software, so if you were getting attacked from the outside and somebody tried to force your firewall to turn off or whatever, then the system would just disconnect or the software would disconnect the system all together from the network, and it would protect it.

Brendan Traw: Great. So can you also show us what this could do for service providers others?

Don Bowden: Sure, let's go ahead and do that. So if we could bring our management

console up on the screen over here. So what I'm going to do first of all is I'm going to do something that's so foreign to a demo guy, that it really pains me to do this, but I'm going to blue screen this system intentionally.

Brendan Traw: Oh. That's ugly. We never like to see that on stage.

Don Bowden: So I'm going to go ahead and start the management console and get things going, and while it's going in and repairing the system, I'll explain to you exactly what I've done here. So give me a couple seconds to get all this set up.

So what I've done is I've started a reboot on the system remotely. Now what happened, when this thing went blue screen, there was a policy running behind the scenes that basically said, "Oops, I have a problem here. I need to contact my service provider." So it opened up the secure, encrypted connection back to the service provider's computer, the service provider went in, looked up the customer information, and the first thing that we did is we went in and established a mount. We actually mounted the disk on our home PC to the system here in the service provider's office. And then we started running a diagnostic OS. Now what this is going to do is it's going to go through, step by step, and reboot the system, and as it reboots the system, it's going to find the problems that occurred here.

Now it can look for things like, you know, corrupted drivers, maybe somebody's introduced a virus onto your system somehow, and in this particular case what we did, when I blue screened the system, I actually overwrote a section of the boot sector, so I corrupted the boot sector. So that's what it's going to do. It's going to go in and find out where the problem is and you can see it's repairing now. Now on this side you can see that I have a hyperterminal screen, so I'm able to see, from my computer in the service provider's office, exactly what's happening on the home computer, so I don't have to be on the phone with the user saying, "Okay, now I need you to go here and click this, or go here and click that." It's all controlled directly from my console right here.

So now we see we've got the message up here that it's ready for reboot, so what I'm going to do is go in and unmount the disk. And I'm going to stop the OS software and just tell it to go through a normal boot. So now when I click reboot here, it's sending down the wire a signal to the home computer to go ahead and reboot. So the problem's been fixed, the system will reboot back up into its repaired state and we've done all this without having to roll a truck or send a technician out.

Brendan Traw: That's terrific. Just one question. Consumers might be concerned about their privacy here. Is the service provider able to arbitrarily come in and look at their system?

Don Bowden: No. The connection has to be established from the home computer out to the service provider. There's no way that the service provider can say, "Yeah, I just want to go look at what this guy's doing today." No way that's going to happen. The connection

has to be initiated from the home computer, and it has to be done with that policy that's set up under AMT.

Brendan Traw: Terrific. Thanks a lot Don.

Don Bowden: You're welcome. Take it easy.

[Applause]

Brendan Traw: Thank you. So with that I'd like to wrap up here. And I hope I've shared with you both our consumer-focused approach that we're bringing to the digital home and in particular the, I think, very exciting content and media experience usage models that are driving everything we're doing with Intel Viiv technology.

I also hope that I've raised some awareness on three sort of critical areas of actually delivering the infrastructure that's needed to enable those experiences and continue to grow those experiences on a going-forward basis, from content delivery to home storage to remote management. And I hope I've inspired many of you to enter into the $1 million design challenge for the Intel Viiv technology PCs with the core processor. And I hope that I've inspired all of you to work on these challenges and help deliver an even more compelling digital home experience to consumers worldwide, working with us in the next few years. Thank you very much. And I can take a few questions.