Geeknet

Slashdot sourceforge free(code):

intel

Parallel Programing: Goals, Skills, Platforms, Markets, Languages

October 2012

Introduction: Extending Moore's Law

Through the first several decades of the microprocessor revolution, software engineers, industry watchers and, eventually, the public at large, all became accustomed to hearing about – and then experiencing – routine, frequent and relatively dramatic improvements in chip performance from doubling of CPU transistor counts and effective clock rates every two years.

Until recently, this meta-phenomenon, articulated by Intel co-founder Gordon E. Moore in a 1965 paper, has remained uncannily accurate. But the now famous "Moore's Law" has also obscured the increasingly abstract relationship over time between transistor counts and CPU performance, especially for the non-IT professional.

Actual performance improvements are the result of a host of technologies built on top of and around the rising number of smaller and smaller transistors, including widening data paths and registers, on-chip caching, instruction prefetch and other technologies that let more work be done in every clock cycle. But though these advances (along with increasing clock rates) handily kept Moore's Law close to accurate through the close of the 20th Century, shortly after the dawn of the new Millennium, the physics and process of chip making hit a long-anticipated wall. Throughput increases for single CPU cores began to stall due to basic physics and process limitations.

Enter Parallelism

Overcoming the problem – and re-engaging with Moore's Law at a higher level of abstraction – meant changing the way chips were engineered and organized in two main ways, both involving parallelization. The first way was to provide on-chip circuitry and instructions for *vectorization*: assembly-language instructions that operate on many pieces of data in parallel. First introduced by Intel on the Pentium 4, this 'Single Instruction, Multiple Data' (SIMD) technology is now in its fourth generation.

The second way was to provide more cores, which led to Intel's introduction first of dual-core CPUs for desktop machines, then quad-core, eight-core and higher core-counts. So-called *multicore* CPUs are now ubiquitous in laptops, desktop computers and servers, and are fast becoming the norm in smartphones, tablets, ultrabooks and other mobile devices. As time goes on, the trend towards multicore is only accelerating, with *many-core* devices like Intel's 50-core Xeon Phi coprocessor board – introduced this year – bringing the potential for supercomputer-like performance down onto PC desktops.

To obtain even more performance – beyond the capabilities of a single machine – various forms of *task parallelism* are also possible. These entail using message-passing and other control methods to coordinate and dispatch work to multiple machines in a cluster or other tightly or loosely coupled federation.

Parallelism and Today's Developer

This has all happened very fast – changing the entire machine landscape in a single decade. Developers most concerned with maximizing application performance now have new hardware paradigms, new techniques of programming, debugging and optimization to master, and a whole new way of thinking about how software works to explore and embrace. University computer science programs offering a full treatment of these emerging themes are still evolving. Organizations such as ACM/IEEE (see http://bit.ly/ICJcpD) only this year are entering final deliberations on their first state-of-the-art-based parallel development curricula. That means the developers most motivated to explore this new technology are often proceeding with little formal training.

Meanwhile, elsewhere in computing, the appearance of ubiquitous multicore and other variants of parallelism has had a more-muted effect, largely due to the ingenuity of chip, system, OS, driver and compiler engineers – all of whom understand the commercial need to deliver performance improvements while minimizing the need for total software overhauls.

As a result of their efforts, much contemporary software can enjoy a parallel performance boost from vectorization-capable, multicore CPUs, coprocessors and other accelerators at relatively low cost of developer time and attention. These 'free or nearly so' improvements are further enhanced by efforts by chipmakers to ensure that upcoming higher performance and/or lower-power-drain multicore CPU microarchitectures are back-compatible with popular existing chips. Thus Intel, for example, has demonstrated its upcoming Haswell low-power processor with on-chip GPU will give existing software an immediate ten percent performance boost, plus double the graphics performance of prior designs, all without even a recompile.

Parallel Programming: Markets, Goals, Challenges

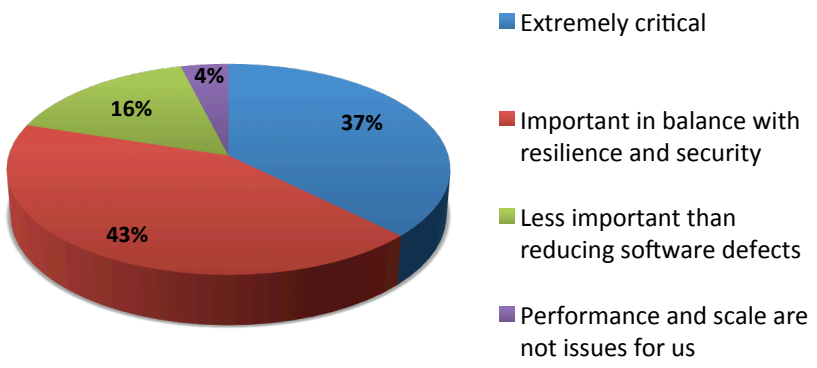
How are software developers and their organizations adapting to new parallel computing architectures? How broad is the market for parallelism – or, more generally, for software acceleration and optimization? What platforms are organizations deploying products on, and for what languages do developers need parallel programming support?

This market report analyzes responses by the Go Parallel developer audience to questions exploring the above topics. The online survey, which garnered 254 responses, was conducted with visitors to http://goparallel.sourceforge.net.

Performance and Scalability

Question 1 explores the need to balance pure software performance and scale with improved resilience, security and overall software quality. Most respondents (43%) said performance and scale were important in balance with other requirements; an almost equal number (37%) viewed performance as most critical.

Question 1: Are performance and scalability – on servers, desktops, coprocessors, mobile, clusters, or supercomputers – critical to your software product's success?



The strong majority emphasis on performance and scale (combined total over 80%) is not surprising. Parallel programming is about *performance*, and in a parallel world, performance is about scale in both directions: Creating code that makes best use of available threads and cores where resources are constrained, but that can scale usefully where more cores are available.

Offering best-possible performance in all conditions provides a competitive advantage in many highly lucrative software categories, for example, in commercial graphics, 3D modeling, CAD and media processing applications, where end users will – in adapting to their own technical, logistical, cost, throughput and workflow requirements – run the same application on a laptop and on a 12-core Xeon-powered workstation, expecting no less than adequate performance from the first and demanding superior performance from the latter.

Though notions of profit and the nature of engagements may differ, much parallel software for medical and biotech, physical simulation, remote-sensing, aeronautics and defense is now evolving under some version of the same constraints and expectations for optimal performance and down/up scalability. And for select markets, introduction of high-end many-core boards like Xeon Phi are likely to create a new top end of expectation for performance.

It's not easy, however, to write optimal parallel software. Program architectures need to be developed – or appropriate patterns selected – to use available cores efficiently, keep them busy, and scale to more cores effectively – ideally offering near-linear performance improvements as core-counts climb. Data needs to be organized in ways that permit fast allocation to waiting worker threads, best use of vector instructions, most efficient use of cache lines. Builds need to be analyzed and tuned to remove unforeseen bottlenecks.

At the same time, the kind of high-value scientific, engineering, medical, aerospace and military applications requiring optimal parallel performance often have equally stringent requirements for correctness – even for mathematical verifiability in some cases. And these requirements may conflict with the nature of how parallel

application components, e.g., threads running independently across multiple cores, work. For example, testing complex, data-dependent applications may require imposing deterministic constraints on order of execution – something quite difficult to do in multi- and many-core systems without imposing significant overheads for synchronization and sequencing.

And indeed, 16% of respondents chose to focus here – on reducing software defects – rather than on performance. That doesn't necessarily mean they're not concerned with performance. In fact, the top three respondent groups – performance-centric, performance/quality balanced, and quality-centric – are probably not strongly distinguished from one another, except in terms of the specific requirements of projects they're undertaking or the nature of institutions in which they work (e.g., software businesses vs. university research teams), these factors in turn influence the degree to which each emphasizes "getting the most bang from the hardware" over "getting the answers right," or vice-versa. Clearly, both are essential and are part of any normal software development process.

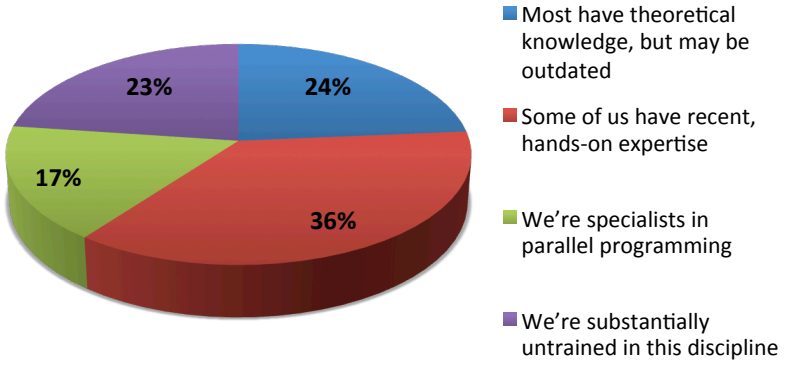
The bottom line, however, is that parallel development is complex. And for many, and perhaps most, programming teams and development scenarios, this intrinsic complexity presents a major business problem: How to organize development to obtain maximum performance (and/or correctness) with minimum expenditure? These days, the best answer can be found in software development tools that serve all facets of process, beginning with the design, through coding, analysis, tuning, optimization and verification.

Intel's Parallel Studio XE 2013 development toolkit, for example, leads in with a package called Intel Advisor XE 2013 – a threading assistant for C, C++, C# and Fortran that helps describe an application's goals, desired deployment platforms and other characteristics. It also analyzes and simulates the software's performance, recommends tests, finds potential flaws pre-implementation, and helps determine which of many parallel development paradigms is best suited for your project. Advisor XE combines performance and correctness analysis, and is equally useful in guiding from-scratch development and helping determine strategy for efficiently parallelizing existing code.

Level of Expertise

Following on this theme, Question 2 probes the level of programming team's existing expertise in multi/many-core parallel programming. Given the speed with which multicore has become ubiquitous, it makes sense only a minority of respondents (17%) classify themselves as multicore specialists. Yet due both to its ubiquity and to competitive pressures to ensure that certain software makes best use of hardware resources -- over half of respondents (54%) report hands-on experience in the discipline.

Question 2: Among the members of your programming team, what's the average level of expertise in programming for multiple threads and cores?



Yet that means that fully still 46% have little or no real experience in multicore. Only about half of these (24%) feel they even have the knowledge needed to assimilate new skills. This doesn't necessarily mean these less-skilled coders are simply more junior. For example, ever-increasing use of computation in the sciences, combined with the lure of 'desktop supercomputing' hardware, suggests that at least some lacking multicore exposure are actually subject-matter experts, steeped in the specifics of valuable algorithms specific to their fields.

The presence of such a large skills gap presents a challenge in training, optimal resource utilization and in productivity enhancement. It confirms the need for flexible tools that simultaneously let multicore specialists tackle complex projects in optimal ways, while also letting less-skilled developers (some of whom may possess unique domain knowledge) build effective parallel applications from scratch or parallelize existing code with good results.

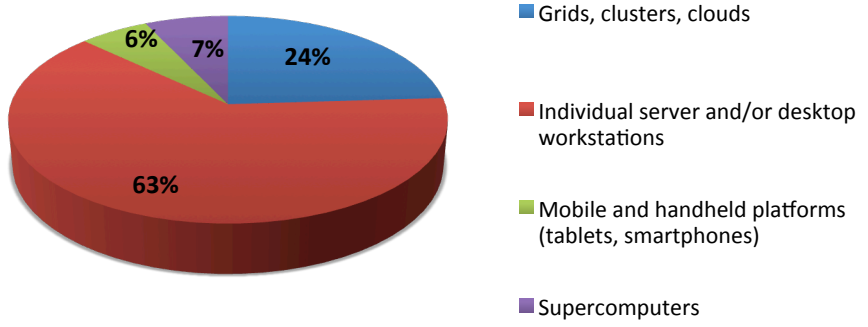
For this reason, top-end parallel development tools like Intel Parallel Studio XE 2013 (and its partner product, Intel Cluster Studio 2013, due later this year), both of which work fluently inside Microsoft Visual Studio, offer developers a range of different parallel development methodologies aimed at different types of projects, programmer skill levels and constraints on time, knowledge and cost.

By far the simplest is Intel Cilk Plus – a terse set of pragmas and a simple array notation for C/C++ enabling rapid and efficient vectorization of loops, and easy creation of custom reducers. More comprehensive (and complementary) is Intel Threading Building Blocks – a library enabling creation of thread-based parallel applications on multicore while abstracting platform details and threading mechanisms. For developers who prefer an open source approach, Parallel Studio XE 2013 also supports OpenMPI, as well as a host of specific, highly optimized parallel libraries for math.

Deployment Platforms

Question 3 asks respondents about deployment platforms. Here, the majority (63%) are focused on developing for desktop computers and servers, while a smaller group (24%) are working with clusters and grids. A minority (6%, but sure to grow) is focused on more mobile platforms such as ultrabooks and performance tablets. Another small group works on supercomputers, presumably including desktop-based.

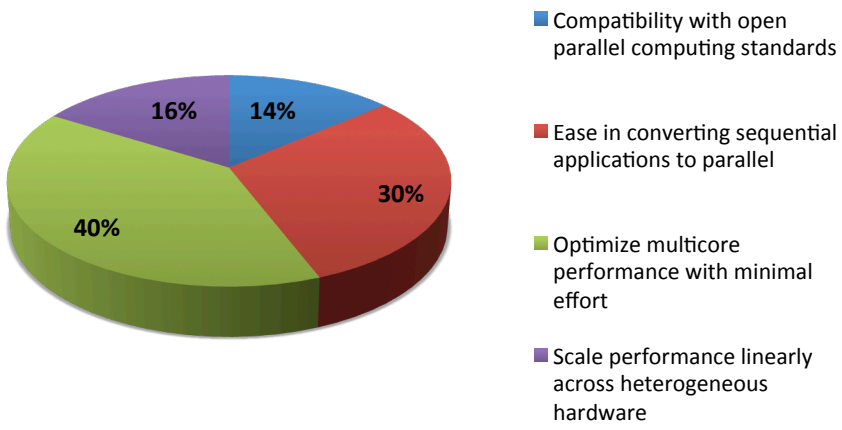
Question 3: What kind of platforms do you deploy applications on?



This mix maps well to the emphasis implicit in Intel's existing developer tool channels – notably Intel Parallel Studio XE 2013, for desktop/workstation software development; and Intel Cluster Studio XE 2013, for developing cluster applications. It's interesting to note that these are essentially the same product, with different deployment models; and that Parallel Studio XE 2013 already includes support for Xeon Phi, which brings supercomputing to the desktop, and which will serve as the compute node in Cray's upcoming Cascade cluster supercomputing line.

Most Important Requirements

Question 4 explores business priorities and requirements for parallel development. The largest group (40%) is pragmatically focused on optimizing multicore performance with least effort. A smaller, but significant group (30%) seeks efficient conversion of convert sequential applications to parallel.



Again, it would seem that Intel's comprehensive support for the full development process neatly attacks both these problems. Optimization is greatly eased in working with Intel's VTune Amplifier XE 2013, an analytic, graphical optimizer and profiler fully integrated with Intel Parallel Studio XE 2013, and offering rapid iteration and comparison of source, object code and live performance data. VTune Amplifier XE lets you find the individual lines in your code that are costing program execution time, visualize issues with their performance, and recode iteratively to fix problems.

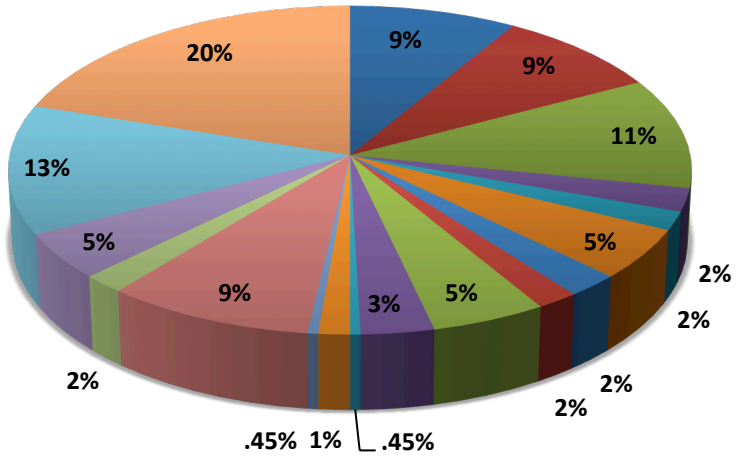
For converting serial applications, an ideal process would begin with Intel Parallel Advisor XE guided analysis and simulation, followed by profiling with VTune Amplifier XE to find opportunities for selective parallelization for maximum effect. Simplified, highly optimized local changes might then be made in Parallel Studio XE 2013, using Cilk Plus semantics for vectorization and Threading Building Blocks, if required, for higher-level, but still abstract creation of task-parallel segments.

Use of abstraction mechanisms like Cilk Plus and TBB speed development and go a long way toward ensuring optimal scalability and portability of code as well. The underlying runtimes are highly optimized for minimum overhead and maximum speed, representing years of continuous R&D by Intel developers, and enabling rapid redeployment of applications optimally on a range of target hardware.

The Proliferation of Parallel

Question 5 explores the range of industries now engaged, or potentially engaged, in parallel software development. What's remarkable here is the diversity, with predictably high numbers of responses from university, consulting, digital content, medical research, environment, resource exploration, engineering and manufacturing, communications and other markets long dependent on high performance computing.

Question 5: What industry does your company/institution belong to?



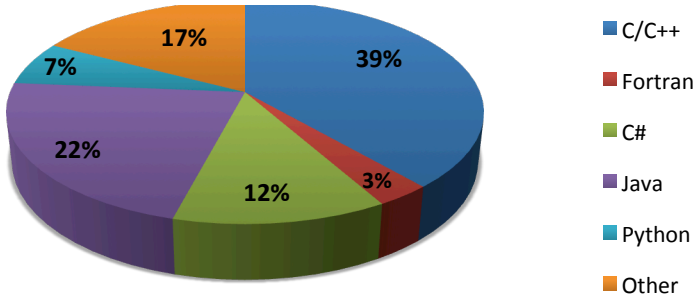
- Computer Aided Engineering (CAE) / Manufacturing
- Digital Content Creation & Distribution
- University/Academic
- Government Lab
- Geosciences and Geo-engineering
- Economics/Financial
- Bio-Sciences / Life Sciences / Healthcare
- Electronic Design Automation (EDA)
- SW Development Tools
- Defense
- Weather
- Chemical Engineering
- Mechanical Design and Drafting
- Communications - Telecom / Datacom Services & Infrastructure
- Retail/Wholesale
- Gaming
- Professional Services/Consulting
- Other

We'd hazard that as true desktop supercomputing performance becomes more generally available due to innovations such as Intel Xeon Phi, and as understanding of parallel development becomes more widespread, penetration into ever-more-diverse businesses will follow as a matter of course.

Languages Used

Finally, question 6 asks about programming language preferences. As one might expect, C/C++-- widely known, highly evolved and high performance – dominates here. But Java and C# figure prominently in survey responses as well, perhaps suggesting growing interest in cluster-parallel development, and perhaps in development efforts targeted at multicore mobile devices. Interestingly, 17% of respondents chose 'Other' – perhaps reflecting growing interest in parallelism-friendly languages like Haskell and Go.

Question 6: Which languages do you use for development?



Conclusion

Statistics from this survey portray a market evolving rapidly, as technology for multicore becomes increasingly widespread and as commercial pressures mount to take better advantage of dramatic performance improvements possible with present and subsequent generations of processors and associated tools.

At present, software developers seem to fall into several fairly distinct groups, well served by Intel's software development product lines. The majority – whether because of limited skill or limited budgets and time - want parallel benefits with minimum fuss, and are drawn to sophisticated but simple tools like Cilk Plus that enable efficient parallelization of existing applications and rapid development of new applications. A more elite market gravitates toward Intel Threading Building Blocks, Cluster Studio, OpenMPI integrations and other more advanced approaches that accommodate the demands of more sophisticated scientific and engineering software development projects.

John Jainschigg is a Geeknet contributing editor, and is CEO of World2Worlds, Inc., a digital agency focused on immersive technology and gaming. John's initial intro to concurrency was via interrupt and re-entrancy programming at the assembler level on Z80 and 68000-based systems. He wrote concurrent, time-critical packet-switching applications on HP-UX RISC machines in the late 1980s, and since then has worked up and down the client-server stack in Java, C++, PHP, and other conventional and scripting languages, and more recently, in task-specific, state-based, radically concurrent languages like LSL.