THE STATE OF

Parallel Programming

The Parallel Programming Landscape

Multicore has gone mainstream — but are developers ready?

An Exclusive Research Report

The proliferation of multicore processors means that software developers must incorporate parallelism into their programming in order to achieve increased application performance. But many programmers are ill-equipped for parallel programming, lacking the requisite training and often relying on primitive development tools. The research shows that better and simpler tools and libraries are needed to help programmers parallelize their code and to debug the complex concurrency bugs that parallelism exposes.

Parallel computing is the primary way that processor manufacturers are dealing with the physical limits of transistor-based processor technology. Multiple processors — or cores — are joined together on a single integrated circuit to provide increased performance and better energy efficiency than using a single processor. Multicore technology is now standard in desktop and laptop computers. Mobile computing devices like smartphones and tablets are also incorporating multicore processors into their designs.

The problem with multicore computing is that software applications no longer automatically benefit from improvements in processor performance the way they did in the past. Those benefits can only be realized by writing applications that expect and take advantage of parallelism.

Sponsored by intel

Dr.Dobb's

In 2006, Saman Amarasinghe, now a computer science professor at MIT, described this as a "looming software crisis"[1] for software developers who write code on platforms that abstract away processor architecture and who therefore don't know how to benefit from parallelism. Fast forward to late 2011, when multicore processors are the norm. Have software developers been able to overcome this "crisis" and incorporate parallelism in their programming? Are the programming tools they use a help or a hindrance? What tools and techniques do programmers need to achieve full performance and correctness on multicore architectures?

UBM TechWeb surveyed Dr. Dobb's readers in October, 2011, to determine how much experience software developers have with parallel programming and what challenges they face in developing software that exploits multicore architectures. What follows is a summary of the state of parallel programming in 2012 based on the survey results and the latest academic research.

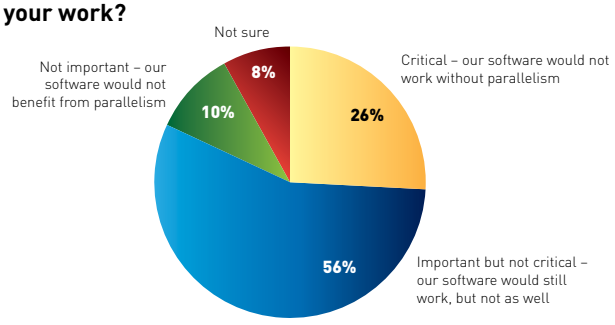
The Parallel Programming Landscape

Crisis or not, awareness about parallel programming is high in the developer community. 81 percent of the developers surveyed report having at least some parallel programming experience. (See Figure 1.) Not only that, but 82 percent of developers now consider parallel programming to be

Figure 1. How much parallel programming experience do you have?

Base: 197 Software engineers/developers
Data: UBM TechWeb Survey of 275 software engineers or managers of development teams, October 2011

Figure 2. How important is parallel programming to your work?

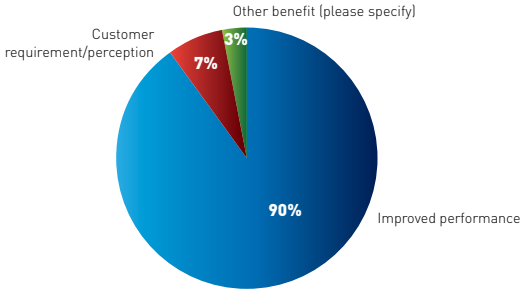


Base: 197 Software engineers/developers
Data: UBM TechWeb Survey of 275 software engineers or managers of development teams, October 2011

important or even critical for the proper functioning of the software they develop. (See Figure 2.) Increased application performance was cited by 90 percent of the managers surveyed as the primary motivation for optimizing applications for parallelism. (See Figure 3.)

The fundamental challenge with multicore architectures is that application performance does not automatically scale to the number of processors. Applications written in a serial (sequential) manner cannot take full advantage of

Figure 3. What was the primary reason for conversion?



Base: 59 directors or managers of development teams who are converting serial applications to parallel code
Data: UBM TechWeb Survey of 275 software engineers or managers of development teams, October 2011

Methodology:

UBM TechWeb conducted a survey on parallel programming for Intel in September and October 2011. The online survey collected data from a total of 275 software engineers or those who manage development teams, nearly all of whom use C/C++ as their primary development language. Results in this paper are based on these 275 qualified respondents unless otherwise noted. The greatest possible margin of error for the total respondent base (N=275) is +/-5.8 percentage points. UBM TechWeb was responsible for all programming and data analysis. These procedures were carried out in strict accordance with standard market research practices.

1. Saman Amarasinghe, "The Looming Software Crisis due to the Multicore Menace," presentation at http://groups.csail.mit.edu/commit/papers/2006/MulticoreMenace.pdf, 2006.

multiple cores without some rewriting. Even multithreaded applications may suffer – concurrency problems that arise when multithreaded code is run simultaneously on separate cores may also negate any perceived performance benefit, or may lead to incorrect behavior and results.

In the academic literature, much of the current research on parallel programming focuses on creating new programming languages with explicit support for parallel constructs or on adapting existing languages with new extensions and libraries. According to Microsoft researcher Sebastian Burckhardt:

> *On multicores, we need programming languages to rise to an abstraction level where correctness issues (such as atomicity violations, data races, and deadlocks) and performance concerns (such as scheduling) are no longer nasty surprises, but become transparent properties of a program … Once we find the right abstractions, they can provide tremendous value, whether they be delivered as libraries, language extensions, or domain-specific languages.*[2]

In practice, however, software developers are most interested in libraries and language extensions, not learning a new language. All but four of the survey respondents use C/C++ as their primary programming language, and the remaining four use Fortran.

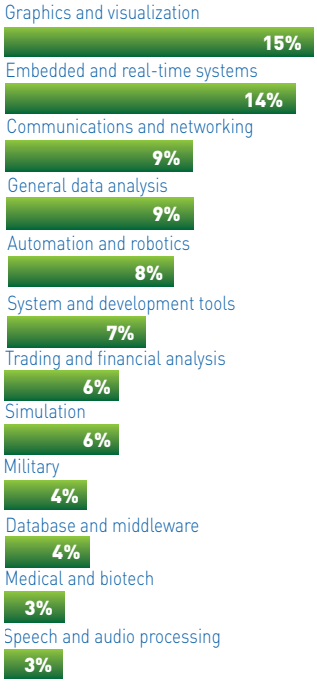
It's interesting to note that parallelism is now used in all types of software. The developers surveyed work on a wide variety of software applications (see Figure 4), including embedded and real-time applications. Ignoring parallel programming issues simply isn't possible anymore. As researchers Mario Leyton and Jose M. Piquer put it:

> *The danger is that parallel programming, which has up to now been reserved for an elite of programmers, will be mandatory for all types of programmers world-wide, along with its complexities and pitfalls.*[3]

Clearly, the survey shows that software developers are incorporating parallelism in their software, but most of those developers would not consider themselves part of the "elite." Computer science professor Ami Marowka further explains the problems that software developers face in using parallelism:

> *Currently, the responsibility to bridge the gap between software and hardware to write better parallel programs may ultimately lie with developers. Many programmers are not up to speed on the latest developments in hardware design. They should study chip architectures to understand how their code can perform better. This is not a desirable situation.*[4]

Figure 4. What kind of software do you develop?



Base: 163 Software engineers/developers
Data: UBM TechWeb Survey of 275 software engineers or managers of development teams, October 2011

One way to fix this situation is to educate software developers about parallelism, which is now happening. Parallel programming courses are now being offered at the undergraduate level by various colleges and universities. Online sites like Dr. Dobb's Go Parallel offer practical continuing education for software developers. New books about parallel programming are also being published.

Education is not enough, however – good programming tools are also needed.

Use of Parallel Programming Tools

Marowka believes that more and better parallel programming tools are needed to help software developers:

> *The lack of multicore programming tools for mainstream developers is perhaps the biggest challenge the industry faces today. ... Parallel programming should be as simple and productive as sequential programming.*[5]

Pedro Fonseca, a researcher in Germany who studies parallel programming issues, agrees with this sentiment:

> *"Many programmers are not prepared to deal with*

2. Sebastian Burckhardt, "Multicore, Manycore, and Cloud Computing: Is a New Programming Language Paradigm Required?" panel discussion reprinted in SPLASH '11 Companion, October 2011.
3. Mario Leyton and Jose M. Piquer, "Skandium: Multi-core Programming with Algorithmic Skeletons", 18th Euromicro ii. Conference on Parallel, Distributed and Networked-Based Processing, 2010.
4. Ami Marowka, "A Study of the Usability of Multicore Threading Tools", *International Journal of Software Engineering and Its Applications*, July 2010.
5. Ibid.

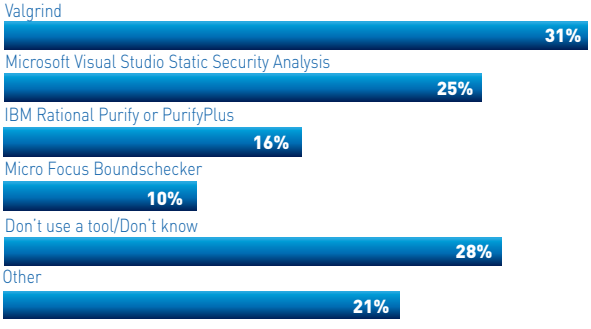
parallelism, i.e., reasoning about parallel programs requires thinking in a very different way. Dealing correctly with parallelism requires training and practice, but having good tools and knowing how to use them can make a huge difference."

There are actually many types of parallel programming tools available today, both commercial and open source, including:

- Parallelizing compilers that generate parallel code either automatically (by analyzing the source code) or using embedded programmer directives
- Parallel-optimized libraries
- Memory checking and profiling tools
- Data race detectors
- Thread interleaving and replay tools

Many survey respondents indicated that they already use one or more of these kinds of tools in their programming, particularly for debugging purposes. For example, 72 percent of respondents use memory checking tools (see Figure 5)

Figure 5. What tools do you use to find memory defects? Check all that apply.



Note: Multiple responses allowed
Data: UBM TechWeb Survey of 275 software engineers or managers of development teams, October 2011

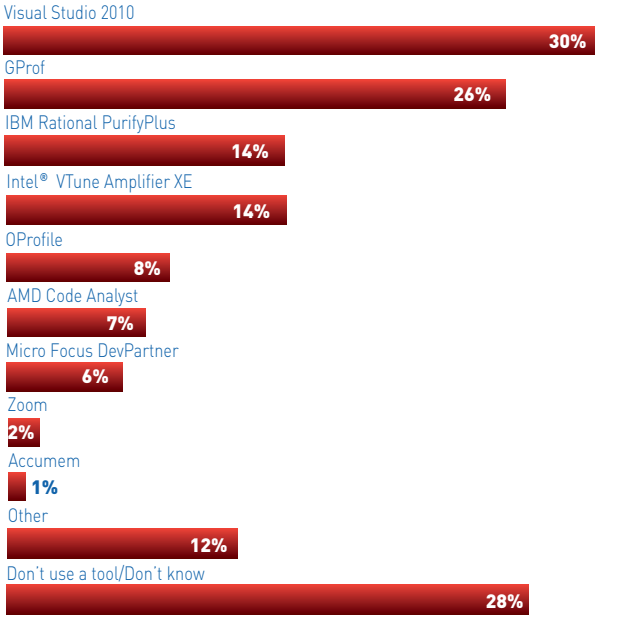
and performance tuning tools (see Figure 6) – not surprising given their long history with serial software development.

But far fewer developers use the other types of parallel programming tools. In fact, a significant number of developers – at least 25 percent for each type of tool surveyed, including memory and performance tuning tools – do not use any software tool at all. Significantly, 66 percent of the developers surveyed do not use any concurrency tool. (See Figure 7.) This is no surprise to Fonseca:

"My perception is that currently developers in general rely mostly on an hoc methods, including simple code inspection, to debug concurrency bugs."

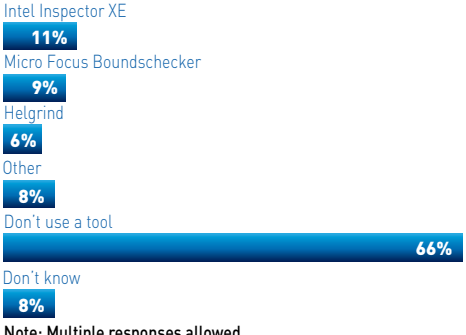
When asked how they find software defects, many respondents agreed with this perception, stating that they

Figure 6. What type of performance tuning tools do you use?



Note: Multiple responses allowed
Data: UBM TechWeb Survey of 275 software engineers or managers of development teams, October 2011

Figure 7. What tools do you use to find threading defects?



Note: Multiple responses allowed
Data: UBM TechWeb Survey of 275 software engineers or managers of development teams, October 2011

use a combination of manual debugging and primitive techniques such as print statements and log analysis to find and fix software bugs.

According to Fonseca, the reasons for this resistance are unclear:

"In part, this may be caused by the fact that some of these tools are not as mature as they could be. Another reason may be that many of these don't come with the usual development tools or do not easily integrate with the development environment (e.g., IDEs)."

The tools may be overly complex, according to Marowka:

The programmer that is using these tools must have the appropriate skills and knowledge. It is impossible to find and resolve deadlocks and data-races without understanding these issues from the theoretical and practical points of view. The user must be an expert in parallel computing and programming.[6]

Cost is likely also a limiting factor, at least for commercial tools, with individual tools often costing hundreds or thousands of dollars per developer.

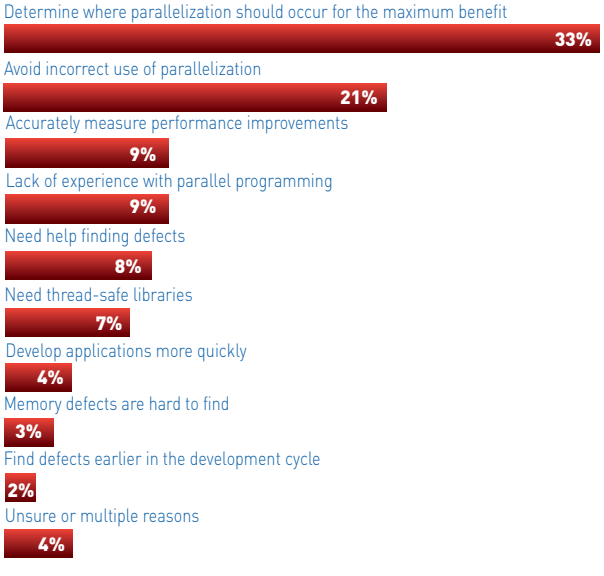
Whether or not they currently use tools for parallel programming, respondents clearly identified a desire for one family of tools: those that would guide them in parallelizing their code.

The Need for Better Parallelization Tools

When asked to rank the parallel programming tool features that would be the most beneficial, 33 percent of participants said that "[determining] where parallelization should occur for the maximum benefit" is the most important, while 21 percent said that "[avoiding] incorrect use of parallelization" is the most important. (See Figure 8.)

In other words, more than half the respondents are interested in tools that would guide them and help them parallelize their code. "It is hard to create good code without the tools to help you identify where things can go wrong and give you advice on how to correct it," said one respondent. Another respondent focused on the correctness issue:

Figure 8. Why will the specified features for parallel programming tools provide you the most benefit?



Note: Multiple responses allowed
Data: UBM TechWeb Survey of 275 software engineers or managers of development teams, October 2011

"Incorrect implementation of parallel code can cause unexpected results and performance issues."

Unfortunately, these kinds of tools are hard to create. Software engineering researchers Hans Vandierendonck and Tom Mens describe the difficulties:

Deciding whether a code excerpt can be parallelized and guaranteeing the correctness of parallelization are complex issues. In fact, performing this analysis based on static code examination is exactly the same problem that an automatic parallelizing compiler faces in discovering parallelizable code – and it is considered unsolvable for general code structures. As such, alternative strategies might be required, such as dynamic or profile-driven program analysis.[7]

Refactoring existing code is particularly challenging, so the researchers recommend that parallelism be part of the design from the start. But the reality is that many software developers today are working on converting existing serial code to parallel code and need help. Said one respondent: "Our software is fairly complex, and it is not readily apparent where parallelizing it would help."

Parallelization tools may not be easy to write, but software developers still need them, and tool vendors should continue to improve them and integrate them more often into existing development environments – anything that makes parallelization simpler will benefit a large cross-section of the developer community.

Note that while survey participants clearly expressed a desire for more and better tools to help them parallelize their code, tools to help debug the threading and concurrency problems arising from parallelization did not receive the same prioritization, although some academics argue these tools are extremely important.

Dealing with Concurrency Problems

Concurrency problems in multithreaded code, parallel or not, have been well documented. Consider the results of an internal survey of Microsoft software developers in 2007, which occurred while single-core machines were still in widespread use:

… about 66 percent of our respondents deal with concurrency issues in one form or another.... Most respondents said that they find the majority of concurrency bugs during system/integration, performance and ad hoc testing.[8]

Concurrency problems in parallel environments are in fact potentially worse than in non-parallel environments

6. Ibid.
7. Hans Vandierendonck and Tom Mens, "Averting the Next Software Crisis," IEEE Computer, April 2011.
8. Patrice Godefroid and Nachiappan Nagappan, "Concurrency at Microsoft – An Exploratory Survey," Microsoft Research, 2008.

because parallel threads of execution run simultaneously. This may expose cross-thread interactions that do not occur in a single-processor environment where only one thread is actually running at any given time.

A 2010 study of concurrency bugs by Fonseca and others found that there was a class of concurrency bugs that was of particular concern, the latent concurrency bug that leads to an application crash long after the bug occurs:

> *Latent concurrency bugs, when triggered, do not become immediately visible to users. Instead, these concurrency bugs first silently corrupt internal data structures, and only potentially much later cause an application failure to become externally visible.*[9]

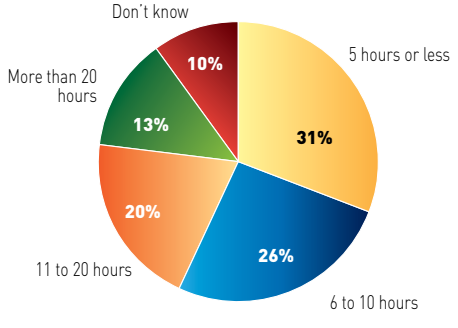
These types of bugs are particularly hard to find because their effects are not immediately apparent.

Approximately 15 percent of the concurrency bugs in the study were latent concurrency bugs, a number that the researchers found surprising:

> *The fraction is large enough that we believe there is value in developing tools that try to recover the internal state of the concurrent application. Performing such a recovery could prevent concurrency bugs from affecting the correct behavior of the application, even after the concurrent requests that cause the error have already been executed and the application state is corrupt.*[10]

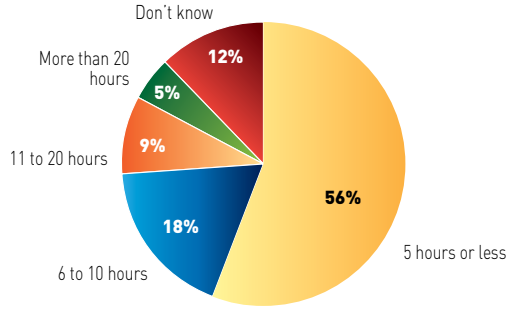
The effects of latent concurrency bugs are perhaps even greater than these numbers suggest, actually. Defects exist that cause applications to fail in ways other than crashing, such as by generating incorrect results. These "Byzantine" bugs are often worse than having the application crash: "Wrong answers are wrong – no matter how fast the computer runs," as one respondent wrote. The researchers found a strong correlation between Byzantine bugs and latent concurrency bugs. In other words, subtle data corruption bugs due to concurrency problems could later cause incorrect application behavior, not just crashes.

Figure 9. How much time do you spend in a month to performance tune your application?



Data: UBM TechWeb Survey of 275 software engineers or managers of development teams, October 2011

Figure 10. How much time do you spend in a month to find a data race or deadlock condition?



Data: UBM TechWeb Survey of 275 software engineers or managers of development teams, October 2011

Developers may not be spending enough time finding and fixing concurrency bugs. More than half of those surveyed spend six hours or more per month performance tuning their applications (see Figure 9), but only 32 percent of them spend that much time finding data race or deadlock bugs. (See Figure 10.) But many concurrency bugs masquerade as memory defects, so developers may actually be spending more time than they think dealing with concurrency issues.

As more sequential code is converted to parallel code, expect the time spent on finding and fixing concurrency bugs to increase, particularly latent bugs. As a result, expect tool vendors to come out with new tools for debugging concurrency problems.

Unlocking the Parallel Advantage

Now that multicore processors are mainstream technology, a majority of software developers are trying to work parallelism into their code in order to boost application performance. Parallel programming is no longer an area of research confined to university laboratories or supercomputer installations, but something that average programmers use in their day-to-day software development, whether or not they feel qualified to do so.

Because parallel programming is complex and error-prone, software developers need good development tools to help them parallelize their code and to find and fix the inevitable bugs, particularly concurrency bugs. Tools already exist, but they are often complex and unintegrated with other tools. Going forward, tool vendors need to create simpler tools that guide developers in their programming and that make it easier to find and fix bugs in parallel code. It's those tools that will truly allow developers to unlock the performance advantage inherent in multicore programming. ■

9. Pedro Fonseca, Cheng Li, Vishal Singhal and Rodrigo Rodrigues, "A Study of the Internal and External Effects of Concurrency Bugs," 2010 IEEE/IFIP International Conference on Dependable Systems and Networks, August 2010.
10. Ibid.