Firm Software Parallelism: Building a measure of how firms will be impacted by the changeover to multicore chips

Neil Thompson

Electrical Engineering and Computer Sciences
University of California at Berkeley

Technical Report No. UCB/EECS-2012-236
http://www.eecs.berkeley.edu/Pubs/TechRpts/2012/EECS-2012-236.html

December 12, 2012

Copyright © 2012, by the author(s).
All rights reserved.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission.

Acknowledgement

I would like to acknowledge the excellent advice and encouragement from my advisor Jim Demmel at the UC Berkeley Parallel Computing Lab. Also in the lab, I'd like to thank Kurt Keutzer, Dave Patterson, and other faculty members for their support and time, as well as Andrew Gearhart and David Sheffield for great feedback and help beta-testing the survey. The participants of the UPCRC ParLab retreats made excellent suggestions that improved this thesis, as did numerous industry contacts that let me interview them. I am grateful to them all. Finally, many thanks to Andrew Fang for excellent research assistance.

Firm Software Parallelism:
Building a measure of how firms will be impacted by the changeover to multicore chips

by
Neil Charles Thompson

Research Project

Submitted to the Department of Electrical Engineering and Computer Sciences, University of California at Berkeley, in partial satisfaction of the requirements for the degree of **Master of Science, Plan II**.

Approval for the Report and Comprehensive Examination:

Committee:

Professor James Demmel
Research Advisor

(Date)

* * * * * * *

Professor David Patterson
Second Reader

(Date)

Abstract

Firm Software Parallelism:

Building a measure of how firms will be impacted by the changeover to multicore chips

by

Neil Charles Thompson

Master of Science in Computer Science

University of California, Berkeley

Professor James Demmel, Research Advisor

The National Research Council has highlighted the importance of the changeover to multicore architectures on software performance. They predict strong economic impacts for users in industry as a result of the change.

This thesis is part of a larger research project that aims to quantify those effects. The main contribution of this thesis is to build a measure of how well firms can take advantage of multicore chips, which is a key ingredient in addressing this important problem. This measure is built up using unique data on firm software usage from the marketing firm Harte Hanks, and combining it with results from the Berkeley Software Parallelism Survey – a survey of parallelism experts run for this thesis. Alternative formulations for creating this parallelism measure are also discussed and evaluated, with the reasons for choosing the main formulation clearly expressed. Finally, this thesis highlights the limitations of this measure and discusses the other on-going initiatives that are underway to address them.

This thesis also makes a supplemental contribution by highlighting a new case study detailing the impact of the changeover to multicore. It also reports additional results from the Berkeley Software Parallelism Survey that may be of interest to those assessing the presence of different types of parallelism or different computational motifs in mainstream software programs.

Acknowledgements

I would like to acknowledge the excellent advice and encouragement from my advisor Jim Demmel at the UC Berkeley Parallel Computing Lab. Also in the lab, I'd like to thank Kurt Keutzer, Dave Patterson, and other faculty members for their support and time, as well as Andrew Gearhart and David Sheffield for great feedback and help beta-testing the survey. The participants of the UPCRC ParLab retreats made excellent suggestions that improved this thesis, as did numerous industry contacts that let me interview them. I am grateful to them all. Finally, many thanks to Andrew Fang for excellent research assistance.

Table of Contents

1 Introduction

The 2011 National Research Council report, *The Future of Computing Performance: Game Over or Next Level?,* makes a compelling case that the shift to multicore microprocessor technology is fundamental and that steps need to be taken to adapt software programming for this new hardware environment. The perils of not doing this, they claim, are that "the development of exciting new applications that drive the computer industry will stall; if such innovation stalls, many other parts of the economy will follow suit."[1] While the report is able to highlight examples in support of this, it cannot offer any economy-wide proof. This is probably for good reason – to the author's knowledge there are no such analyses to report. The absence of these studies is important, because such results would change the argument from 'it is logical that', to 'it has been shown that' – a difference which would make the report's recommendations more compelling to a public policy audience.

This thesis is part of a larger research project designed to estimate these economy-wide effects. That project has many parts, interweaving computer science, economics, and statistics. The contribution of this thesis is to clarify the computer science underpinnings of the project and to develop a measure of how much a firm will be impacted by the switch to multicore processors. Because of the enormous complexity in how firms implement and use Information Technology (I.T.), this will not be a perfect measure. The goal is instead to provide a principled starting place for this analysis, a reasonable first version of that measure, and an understanding of how future work can strengthen it.

The organization of this thesis is as follows. Section 2 discusses the shift from chip speed-ups to multicore, and discusses why this change is important. Section 3 outlines the methodology and data that are used to build the firm-level measure of software parallelism – the metric that will be used to estimate the impact of multicore for that firm. Section 4 reports the results of this aggregation to the firm level and discusses the strengths and weaknesses of the measure. Section 5 outlines how, in other work, this metric is used to address the broader public policy question, and reports the preliminary results from that analysis. And finally, Section 6 concludes the thesis.

[1] NRC (2011)

2 The Shift to Multicore

By the mid-2000s heat generated from increased microprocessor clock speeds was creating a heat dissipation problem within microchips, as shown in Figure 1.

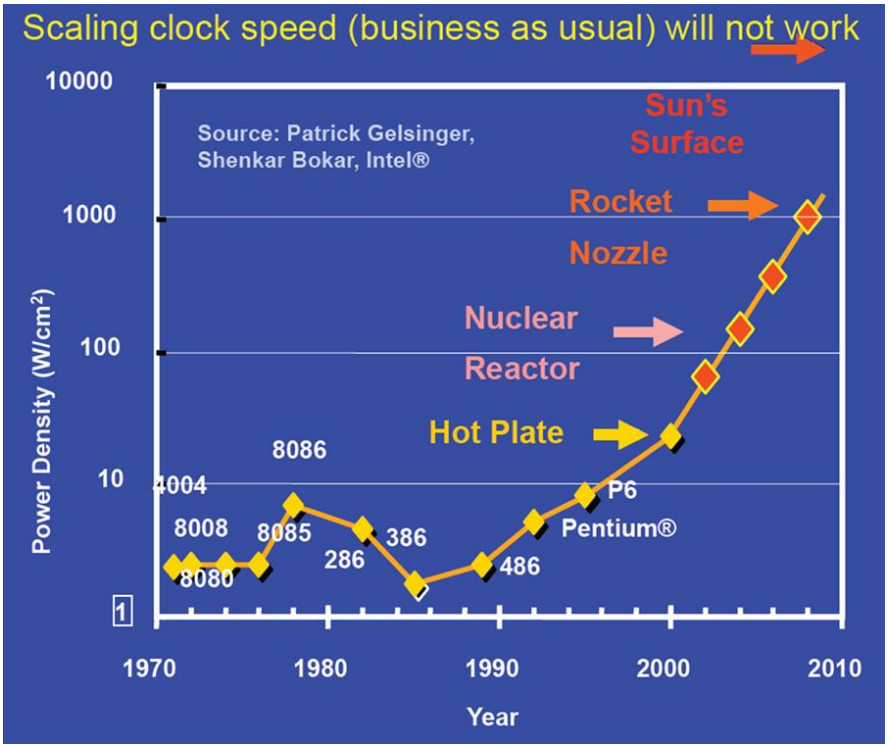


Figure 1: Intel power density projection

This issue was so severe that in 2004 Intel announced that it would abandon future Pentium 4 releases, and that instead it would "dedicate resources to pushing dual-core processors."[2]

This changeover ended a forty-year trend of exponentially rising clock-speeds as a result of Moore's Law driven improvements.[3] The graph below, based on data from Olukotun et al. (2012), shows the progress of transistor density increases (Moore's Law – blue line), clock-speed improvements (red line), and number of processors per computer chip ('Cores' – green line).[4]

[2] Regan (2004).

[3] For each doubling of microprocessor transistor density, a gain of 1.6 or 1.7 was achieved in clock-speed (Brock (2006), p103).

[4] Data from Olukotun et al. (2012), graph from Berkeley CS 294-73 class slides.

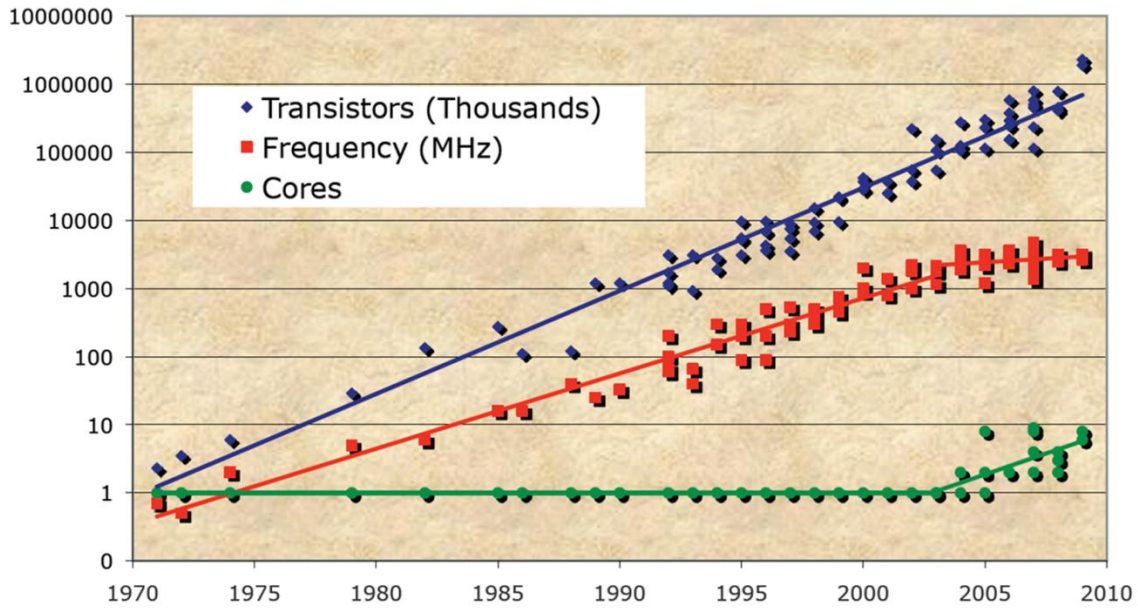


Figure 2: Chip speeds, cores, and transistor densities over time

The transition in 2004/2005, with the end of clock-speed improvements and the rise of multicore chips, is the focus of this thesis. Patrick Gelsinger, Senior VP at Intel and Chief Architect for the 486 microprocessor has called this shift to multiple cores the single greatest shift to date in microprocessor architecture.[5] The NRC report explains the importance of this change: before the changeover "a new computer could run new applications, and the existing applications would run faster," but afterwards "the only foreseeable way to continue advancing performance is to match parallel hardware with parallel software."[6] Put another way, all software that was computationally constrained benefited from clock speed improvements, whereas only those that could avail themselves of parallelism benefited afterwards.

The impact that this changeover had can be seen in the following graph from Hennessey and Patterson (2011) on the performance on the SPECint benchmark:

[5] Brock (20060, pp102-104.

[6] NRC (2011), pp 1-2.

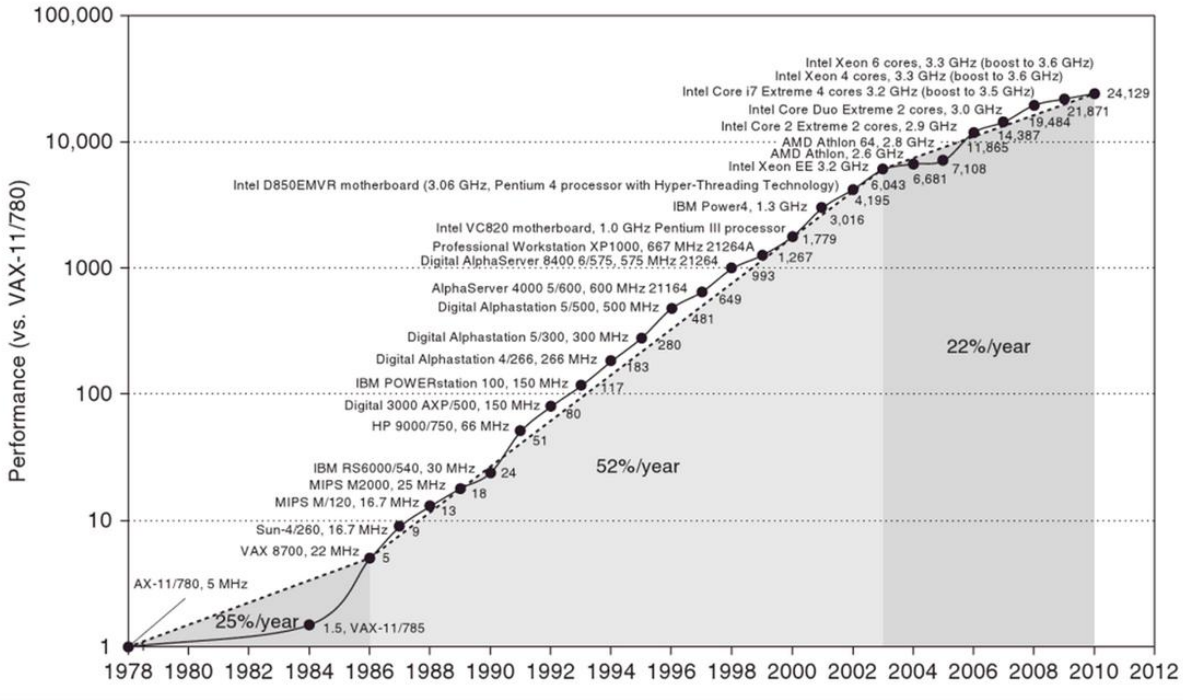


Figure 3: Performance on SPEC Benchmarks

As this figure shows, when processor clock-speed growth stagnates, improvements in SPECint performance diminish. A similar trend can be seen in the performance of SPECfp.[7]

Since these benchmarks are intended to reflect computationally-relevant workloads, the decrease in the rate of their improvement implies a decrease in the rate of improvement of software used by firms. This is important for users in industry because firms take advantage of software performance improvements to make themselves more productive.

An example of this is Dreamworks, the company which produces the *Shrek* movies. They have coined the expression "Shrek's Law" to reflect the doubling of computing power that they need with each subsequent movie release to make their animation improvements.[8] After the multicore changeover, Dreamworks found that "single threaded CPU performance is no longer increasing at previous rates, we must find alternative means to improve rig performance…[and so we focused] on multithreading our animation software…to take advantage of multicore CPU architectures."[9]

[7] NRC (2011), p157.

[8] The gap between Shrek movies is ~2-3 years.

[9] Watt et al., (2012).

This evidence suggests that the impact of the multicore switchover should already be visible, with firms using parallelized software getting more benefits, and firms using less parallelized software getting fewer benefits. The next section describes a methodology for doing this analysis.

3 Methodology and Data

Constructing a measure of firm-level software parallelism requires three components. The first is data on the software that firms are using. The second is an understanding of how these programs' performances will be impacted by multicore chips. The third is a way to aggregate these individual software impacts to a firm-level effect. For firms where all three components of this methodology can be obtained, an estimate of firm-level software parallelism can be obtained. This section outlines the data and methodology used to characterize each of these components.

3.1 Firm software data

Rarely is it possible to observe the software that firms are using. Companies seldom announce this to the public, and it seems not to be part of any public reporting done to government or other regulatory bodies. Indeed, to the author's knowledge, there is only a single dataset that tracks the specific software used by many firms. It is used in this analysis.

Market research firm Harte Hanks collects data on I.T. usage at firms using surveys.[10] These surveys are conducted by phone, with companies self-reporting their usage. The data are gathered to enable I.T. firms, who are clients of Harte Hanks, to target firms for their marketing and sales efforts. Harte Hanks describes their data as follows:

> *"The CI Technology Database is the most comprehensive source of technology information in the world today. Covering North and South America as well as Europe, more than 400,000 payrolled establishments (sites) have received detailed technology interviews. Nearly all of the information in the database is collected via telephone interviews. More than 65,000 interviews are completed each month by 300 tele-researchers with key technology contacts at the largest technology installations in North America, Latin America, and Europe."*[11]

[10] A (somewhat dated) description of the European data is available for public consumption in Harte Hanks (2006). See, in particular, the Product Table and the Product Specification Table.

[11] Harte Hanks (2002).

Of these markets, only the Swedish data were available for this research.

Marketing materials from Harte Hanks, albeit for their U.S. data, provide additional details on the firms covered by this survey.[12] It shows that, of the 10 million U.S. establishments with payrolls, they survey 270 thousand (~2.7%). Of these, they cover 84% of establishments with 1000+ employees, 83% of establishments with 500-999 employees, 62% of those with 100-499 employees; 21% of those with 50-99 employees, and 1% of those with fewer than 50 employees. Thus their sample represents a view that is biased towards larger companies. Considering the absolute number of surveys conducted implies that the median firm in their sample would have ~100 employees.

These surveys are done in great detail and provide information on the following areas of firm technology usage:

- Computer System / Servers
- Local Area Network
- Network Connection Hardware
- Network Connection Services
- Operating Systems
- Personal Computers
- Server Operating Systems
- Software

This work focuses on the software data collected, in particular: Operating Systems, Server Operating Systems, and Software. Taken together, these provide a detailed glimpse of the software that firms are using. Table 1 shows a list of the software classes used in this thesis. These represent the most widely used software classes from the Harte Hanks survey. Examples and links for more information on these software classes are also provided.

[12] Harte Hanks (2002).

Software Class	Example	Link to Description of the Software Class
antivirus	Norton Antivirus	http://en.wikipedia.org/wiki/Anti-virus_software
compiler	C++ compiler, Java compiler	http://en.wikipedia.org/wiki/Compiler
customer relationship mgt	SAP CRM	http://en.wikipedia.org/wiki/Customer_relationship_management
database	SQL Server	http://en.wikipedia.org/wiki/Database_management_system
email	Microsoft Outlook	http://en.wikipedia.org/wiki/Email_client
finance & accounting	SAP Financial Accounting (SAP FI)	http://en.wikipedia.org/wiki/Accounting_software
human resources mgt	Oracle HRM	http://en.wikipedia.org/wiki/Human_resource_management_system
network mgt	HP Openview	http://en.wikipedia.org/wiki/Systems_management
network operating systems (network os)	Windows Server	http://en.wikipedia.org/wiki/Network_operating_system
operating systems (os)	Microsoft Windows	http://en.wikipedia.org/wiki/Operating_systems
presentation	Microsoft Powerpoint	http://en.wikipedia.org/wiki/Presentation_software
resource planning system	SAP MRP	http://en.wikipedia.org/wiki/Material_requirements_planning
spreadsheet	Microsoft Excel	http://en.wikipedia.org/wiki/Spreadsheet
web browser	Internet Explorer, Firefox	http://en.wikipedia.org/wiki/Web_browser
word processing	Microsoft Word	http://en.wikipedia.org/wiki/Word_Processing

Table 1: List of Software Classes

The data are a panel, showing ~6,600 Swedish firms from 2001 to 2007. To give a flavor of the data, Figure 4 reports the percentage of firms using various types of software. Because not all firms are sampled each year, the results reflect the most-recent survey as of 2003.

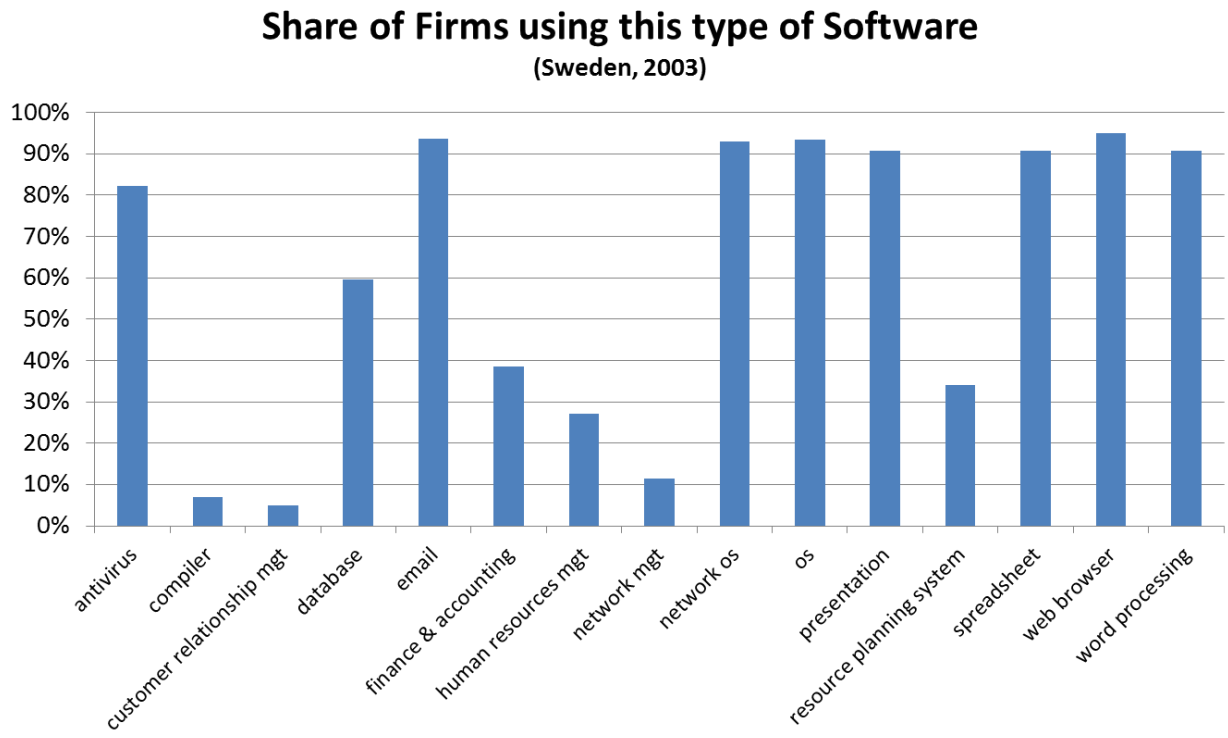


Figure 4: Swedish firm software usage

This remarkable data from Harte Hanks allows the usage of software to be observed, in detail, at the firm-level over time. As a result, it directly addresses the first of the three components needed for building the firm software parallelism metric.

3.2 Software Parallelism

Assessing the parallelism in software is difficult. For any program, there will be many possible workloads and many possible usages. Some of these will take better advantage of parallelism than others. Nonetheless, it is straightforward to consider this a distribution, for which there exists a mean, and therefore where it is meaningful to talk about the average parallelism of that piece of software. While the term 'average' will often be dropped in the discussion that follows, it is that concept which should be kept in mind.

Even having created a reasonable definition of software parallelism, actually estimating that parallelism is difficult. The following sub-sections explore a range of possibilities for assessing software parallelism. They also comment on the data availability and reliability of those possibilities.

3.2.1 Software Package Benchmarks

For the purpose of this analysis, the ideal data would be periodic assessments of the performance of whole software packages for established sets of commercially-important tasks over time. An example of this might be the testing of Excel across a broad range of business-relevant tasks, which were then aggregated to a single value based on their frequency and importance in usage for firms. If such an analysis were to be carried out each year on the microchips on the market, it would provide an excellent estimate of the benefit that Excel was getting from the hardware.

Sadly, the author has found no evidence that such data exists publically. Discussions with hardware / software manufacturers have hinted that this data may be available, in whole or in part, inside those organizations. Negotiations to get access to that data have not yet been successful, but continued attempts are ongoing.

3.2.2 Company Announcements

A second source of data on software parallelism comes from the announcements that company's make when they release new products. For example, the following text comes from Symantec's Jan 2005 announcement:[13]

> *"Symantec AntiVirus 10.x and Symantec Endpoint Protection 11.x use a new scan engine that supports multiple threads. This feature is primarily recommended for use on servers, and is only available on Windows platforms. Performance gains from this feature depend on the computer system setup and multithreaded settings."*

While this clearly provides a helpful notice of a qualitative change in how the software works, it fails to shed light on the quantitative impact of that change. As such, it would be particularly difficult to use this type of data to compare the impact of multithreading antivirus software, versus, for example, a change in multithreading for a database. Despite this shortcoming, this data is good for comparisons within software classes. This is particularly valuable since those comparisons are a shortcoming of the other methods discussed in Sections 3.2.3 and 3.2.4. As such, this data is being gathered for a follow-on piece of analysis.

[13] Symantec (2005).

3.2.3 Surveying Parallel Computing Experts

A third method of evaluating software parallelism is to survey experts that do work in software parallelism, in this case: professors, graduate students and industry professionals. This survey, the Berkeley Software Parallelism Survey, was carried out using the *Qualtrics* online platform. The survey went to ~330 people from UC Berkeley, Stanford, and the University of Illinois, as well as industry contracts from the U.C. Berkeley Parallel Computing Lab (ParLab).[14] Of these 80 (24%) completed the first set of questions, and 49 (15%) completed through to the final question (although, importantly, respondents only answered about classes of software about which they were knowledgeable). Of those that reached the end, 14% were faculty, 37% were students, 39% were from industry, and 10% self-identified as 'other' (e.g. post-doc).

Prior to distribution, the survey was beta tested by Jim Demmel, Andrew Gearhart, and David Sheffield from ParLab, as well as representatives from Microsoft and Intel. The data presented below represent the first wave of this survey. Additional survey waves are planned for the near future.

All the questions from the full survey can be viewed at http://faculty.haas.berkeley.edu/neil_thompson/Berkeley_Software_Parallelism_Survey/Berkeley_Software_Parallelism_Survey.pdf[15]

As part of the Berkeley Software Parallelism Survey, respondents were asked the following questions (shown here for the database management systems category):[16]

[14] Including people at the following institutions: CNMAT, IBM, Intel, International Computer Science Institute, Lawrence Berkeley National Lab, Microsoft, National Instruments, NEC Labs, Nokia, NVIDIA, Oracle, Samsung, and Xilinx.

[15] The formatting and page breaks on the real survey are suppressed to get this output.

[16] Survey recipients were also asked about the parallelism in individual pieces of software, but initial results suggest that few participants were able to answer the question at that level of depth. This question is shown in an Appendix, section Appendix A.

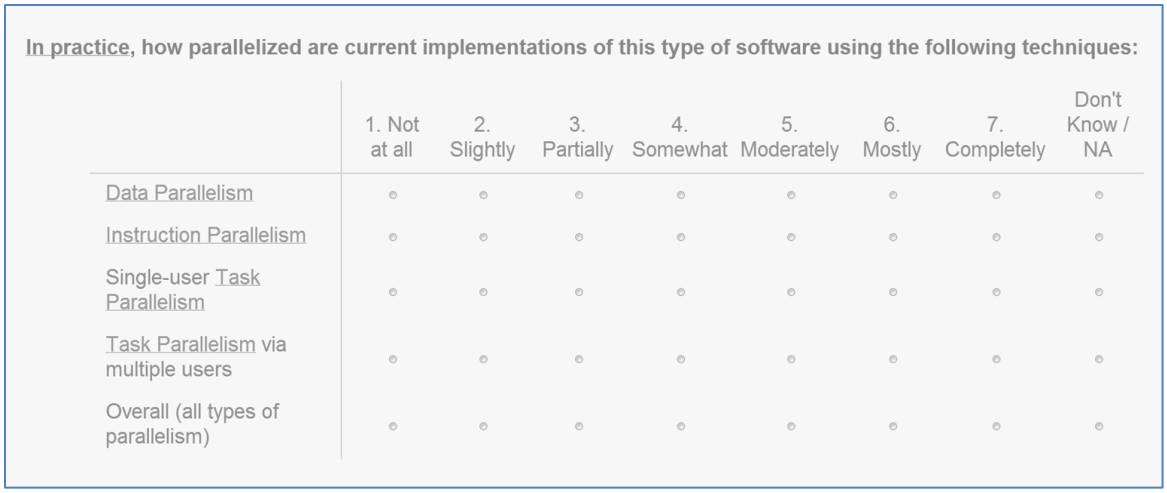


Figure 5: "In Practice" question from Berkeley Software Parallelism Survey

These questions were asked for each of the software categories from the Harte Hanks data. As shown above, the principal question was "In practice, how parallelized are current implementations of this type of software...," as shown in Figure 5. Respondents were also asked a similar question, beginning "In theory, how parallelizable is this type of software..." Figure 6 shows the ratings for each of the group. Only the overall level of parallelism is shown here, the component results are reported in the Appendix, Section Appendix B. For ease of interpretation, the survey ratings from 1-7 have been mapped to a 0-1 range.[17]

[17] In detail: Likert scale of 1-7 is converted to a 0-100% scale, with "not at all" and "completely" referring to the end-points, and an assumption of even-spacing between items on the scale. For the graph, the total number of respondents for some questions is small, both because of the overall response rate, and also because – to keep the survey length manageable – not every respondent answers every question. Hopefully subsequent waves of the survey will bolster small sample sizes. Current sample sizes can be seen in the Appendix, Section Appendix A. See, for example, http://en.wikipedia.org/wiki/Likert_scale, for a description of the Likert scale.

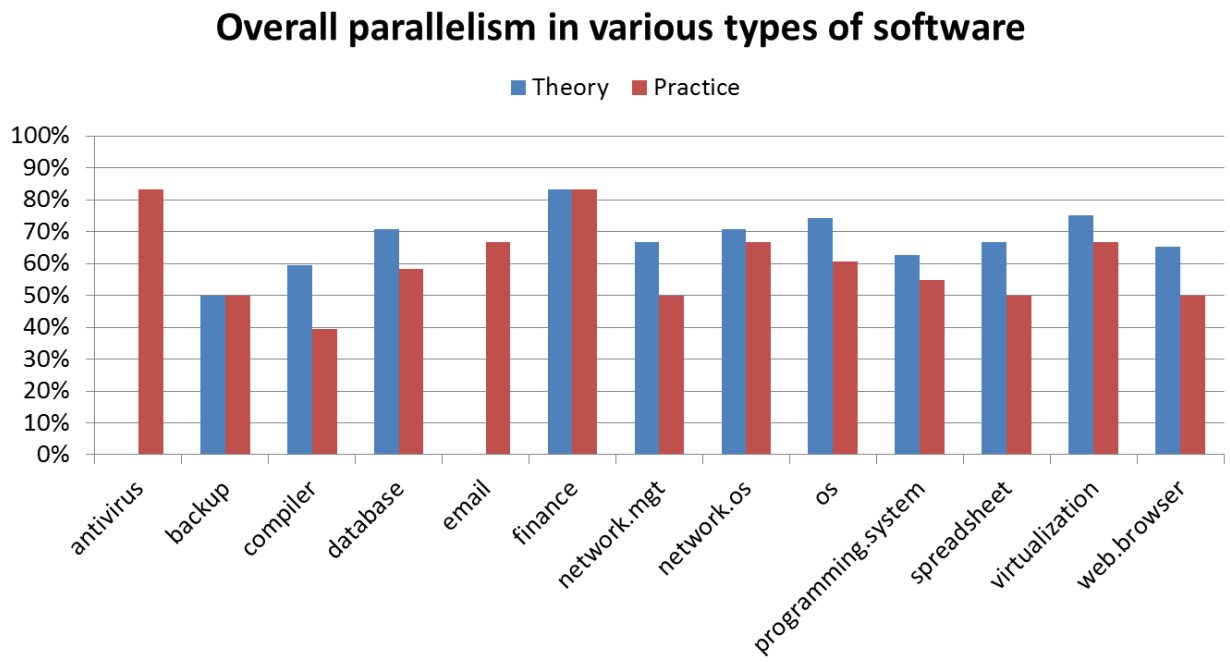


Figure 6: Berkeley Software Parallelism Survey Results for Overall Parallelism

As one would expect, the ratings for "in practice" are all lower than "in theory." For antivirus and email, no responses for "in theory" are shown, because no respondents answered that question – perhaps because it is difficult to know / judge. Of these two sets of responses, the "in practice" is the measure used here, since it reflects the actual gains that would be expected by firms, rather than the gain they could have gotten in theory.

3.2.4 Assessing Parallelism through motifs

Another way of determining the parallelizability of software is to consider the underlying algorithms used in the software, and to consider those algorithms' level of parallelizability. Below is a list of the U.C. Berkeley Computational motifs, a set of 12-13 patterns that recur frequently in programming. These are used as the building blocks for assessing software parallelism:

Computational Motif	Description (from: Berkeley View wiki) [18]
1. Dense Linear Algebra	These are the classic vector and matrix operations, traditionally divided into Level 1 (vector/vector), Level 2 (matrix/vector), and Level 3 (matrix/matrix) operations. Data is typically laid out as a contiguous array and computations on elements, rows, columns, or matrix blocks are the norm.
2. Sparse Linear Algebra	Sparse matrix algorithms are used when input matrices have such a large number of zero entries that it becomes advantageous, for storage or efficiency reasons, to "squeeze" them out of the matrix representation. Compressed data structures, keeping only the non-zero entries and their indices, are the norm here.
3. Structured Grids	Data is arranged in a regular multidimensional grid (most commonly 2D or 3D, sometimes 4D, but rarely higher). Computation proceeds as a sequence of grid update steps. At each step, all points are updated using values from a small neighborhood around each point.
4. Unstructured Grids	Many problems can be described in the form of updates on an irregular mesh or grid, with each grid element being updated from neighboring grid elements.
5. Spectral Methods	Data is operated on in the spectral domain, often transformed from either a temporal or spatial domain. During a transformation, spectral methods typically use multiple stages, where the dependencies within a stage for a set of butterfly patterns (eg. Cooley-Tukey Fast Fourier Transform or the many variants of Wavelet transforms).
6. Particle Methods / N-body	N-Body methods involve calculations that depend on interactions between many discrete points. There are many variations of the basic concept: in particle-particle methods, every point depends on all others, leading to an O(N^2) calculation.
7. Monte Carlo Methods	[see MapReduce below, of which this is a special case]

Table 2: Description of Berkeley Computational Motifs 1-7

[18] Berkeley View (2008). Further information is available at OPL Working Group (2012).

Computational Motif	Description (from: Berkeley View wiki) [19]
8. Combinatorial Logic	Combinational logic describes many simple, yet important functions which exploit bit-level parallelism to achieve high throughput. Workloads dominated by combinational logic computations generally involve performing simple operations on very large amounts of data. For example, computing checksums or CRCs is critical to network processing and ensuring data archive integrity, and can be accelerated with combinatorial logic. Performing simple Boolean operations such as and, or and xor on large data sets is important for applications such as RAID, and falls into the combinational logic category.
9. Finite State Machines	Computation represented as a finite state machine (FSM) is described by an interconnected set of states.
10. Backtrack / Branch & Bound	Branch and bound algorithms are effective for solving various search and global optimization problems. The goal in such problems is to search a very large space to find a globally optimal solution. Since the search space is intractably large, some implicit method is required in order to rule out regions of the search space that contain no interesting solutions. Branch and bound algorithms work by the divide and conquer principle: the search space is subdivided into smaller subregions (this subdivision is referred to as branching), and bounds are found on all the solutions contained in each subregion under consideration.
11. Graph Algorithms	Graph Traversal applications must traverse a number of objects and examine characteristics of those objects. Applications typically involve indirect lookups and little computation.
12. Dynamic programming	Dynamic programming is an algorithmic technique that compute[s] solutions by solving simpler overlapping subproblems. It is particularly applicable for optimization problems where the optimal result for a problem is built up from the optimal result for the subproblems.
13. MapReduce	This dwarf was originally called "Monte Carlo", after the technique of using statistical methods based on repeated random trials. The patterns defined by the programming model MapReduce are a more general version of the same idea: repeated independent execution of a function, with results aggregated at the end. Nearly no communication is required between processes.

Table 3: Description of Berkeley Computational Motifs 8-13

To use these motifs to assess the performance of software it is important to understand both the amount of parallelism in each of the motifs, and the relative usage of the motifs in each type of program.

A starting place for assessing the parallelism in each motif is the work by David Sheffield and Kurt Keutzer from the U.C. Berkeley Parallel Computing Lab. They assign to each motif a level of data parallelism, task level parallelism, and instruction level parallelism – three important aspects of parallelism that can be used to improve performance. Table 4

[19] Berkeley View (2008). Further information is available at OPL Working Group (2012).

shows basic definitions for these types of parallelism; for more detail see OPL Working Group (2012).

	Definition
Data Parallelism	Simultaneous operations across large sets of data.[20]
Task Parallelism	Simultaneous execution of multiple threads of control.[21]
Instruction Parallelism	Overlapping the execution of instructions by the processor.[22]

Table 4: Definitions of different types of parallelism

Figure 7 shows the level of parallelism available in each motif (cool colors imply little parallelism and hot colors imply lots of parallelism).[23]

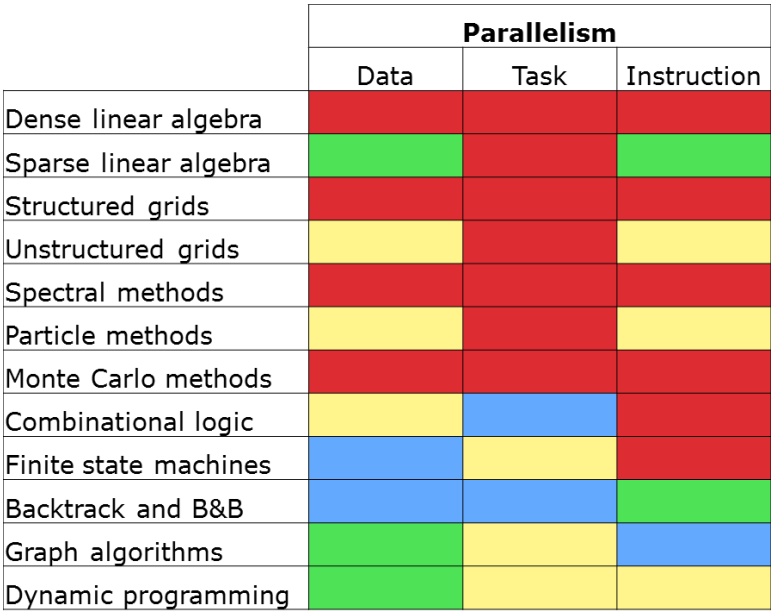


Figure 7: Sheffield-Keutzer Parallelism Chart

While this provides a four-stepped division of parallelism, a more-finely quantified version of the parallelism would be desirable. This finer detail is precisely what IBM researchers Cabezas and Stanley-Marbell (2011) do, measuring the level of task-level and instruction-level parallelism in software. They do this by mapping the motifs to applications from the SPEC CPU2000, MiBench, and Phoenix benchmarks that are dominated by those motifs.

[20] Hillis and Steele (1986).

[21] Hillis and Steele (1986).

[22] Hennessy and Patterson (2007), p66.

[23] The formatting of this chart, but not the content, has been simplified for presentation here. Notice also that this chart includes only 12 motifs, reflecting the difficulty in classification at the margin.

For example, they assign JPEG encoding and decoding to dense linear algebra (although also to structured grids), and gzip to finite state machines. Figure 8 shows the motifs (a.k.a. 'dwarfs') tested by Cabezas and Stanley-Marbell, as well as their short-form abbreviation, and the application they used for testing.

Constituent/ Dominant Dwarf		Applications
1: Dense Linear Algebra	DLA	JPEG[b](enc., dec.), kmeans[c], pca[c]
2: Sparse Linear Algebra	SLA	basicmath[b], bitcount[b], matrixmultiply[c]
3: Spectral Methods	SM	lame[b], FFT[b], IFFT[b]
4: N-Body Methods	NB	None
5: Structured Grids	SG	JPEG[b](enc., dec.)
6: Unstructured Grids	UG	equake[a]
7: MapReduce	MR	mesa[a]
8: Combinational Logic	CL	rijndael[b](encode, decode), sha[b]
9: Graph Traversal	GT	qsort[b], Dijkstra[b]
10: Dyn. Programming	DP	None
11: Backtrack Branch+Bound	BT	vpr[a], mcf[a], art[a], susan[b]
12: Graphical Models	GM	None
13: Finite State Machine	FSM	gzip[a], gcc[a], parser[a], stringsearch[b]

Figure 8: Applications Tested by Cabezas and Stanley-Marbell

Figure 9 shows their findings for the parallelism available in each of these programs. The circles are identified with their abbreviation from above.

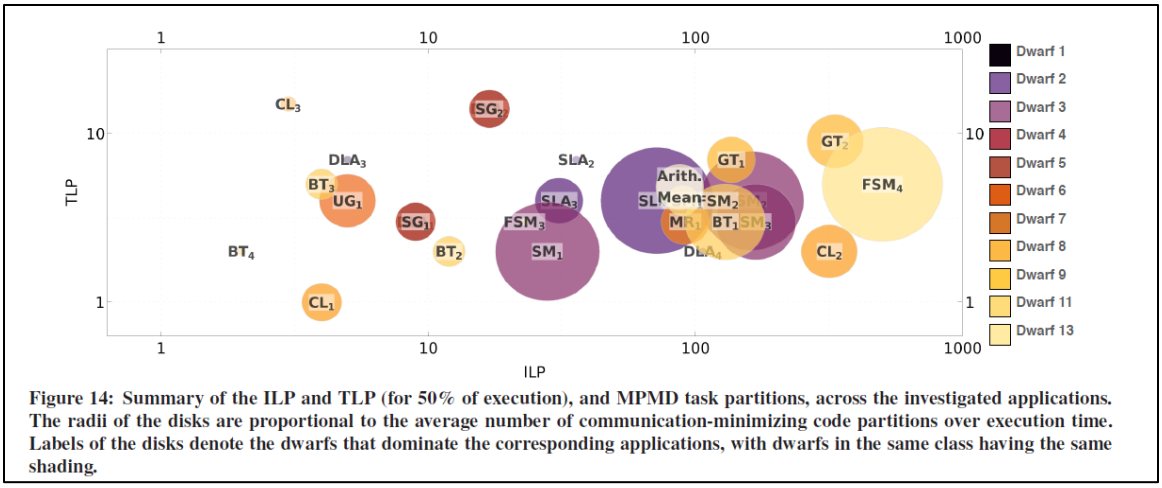


Figure 14: Summary of the ILP and TLP (for 50% of execution), and MPMD task partitions, across the investigated applications. The radii of the disks are proportional to the average number of communication-minimizing code partitions over execution time. Labels of the disks denote the dwarfs that dominate the corresponding applications, with dwarfs in the same class having the same shading.

Figure 9: Cabezas-Stanley-Marbell Graph of ILP and TLP in Berkeley Motifs

This shows, for example, that the Finite State Machine motif (FSM4) has instruction-level parallelism (ILP) of between 100 and 1000 and task-level parallelism of 1 to 10. This is measured at the median time point – meaning that for 50% of the time there is greater than this level of parallelism available, and for 50% of the time there is less parallelism available.

This graphs also highlights that the motifs do not map perfectly onto levels of parallelism. For example, the Combinatorial Logic (CL2) bubble is placed near the FSM4 bubble, but another application that also uses Combinatorial Logic (CL1) has a very different level of instruction-level parallelism, and a third (CL3) has a different level of Task-Level parallelism.

Figure 10 and Figure 11 illustrate the extent of this difference, by showing the mean and standard deviation across the programs embodying each motif.[24]

[24] Many thanks to Philip Stanley-Marbell for sharing the underlying data that allowed for these calculations.

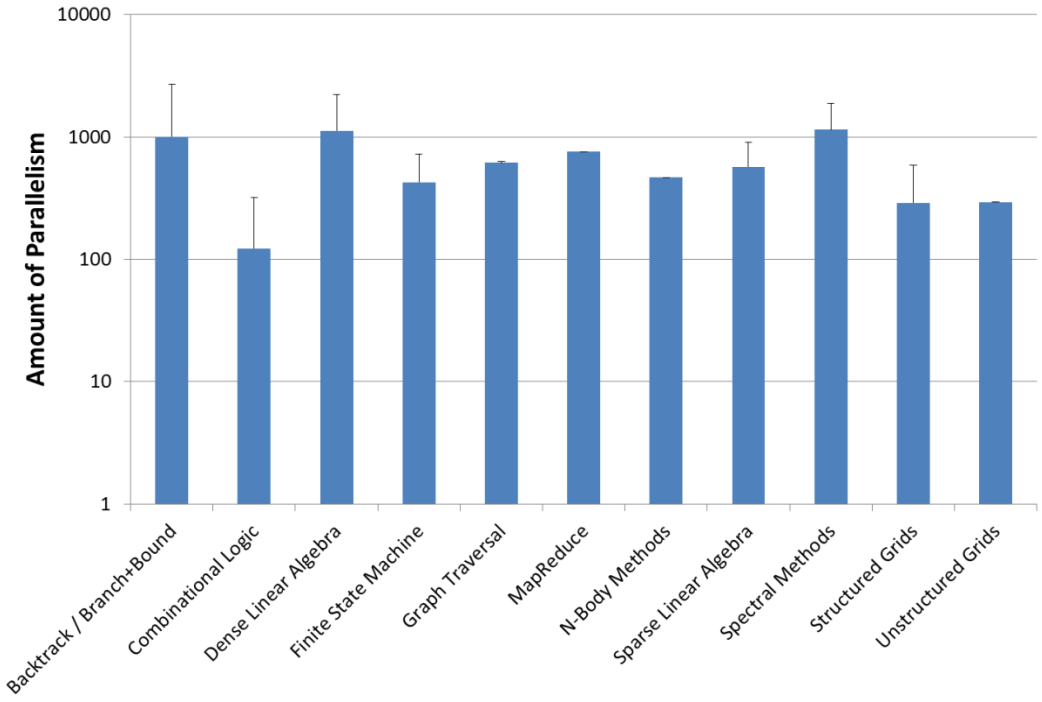


Figure 10: Mean instruction-level parallelism in motifs

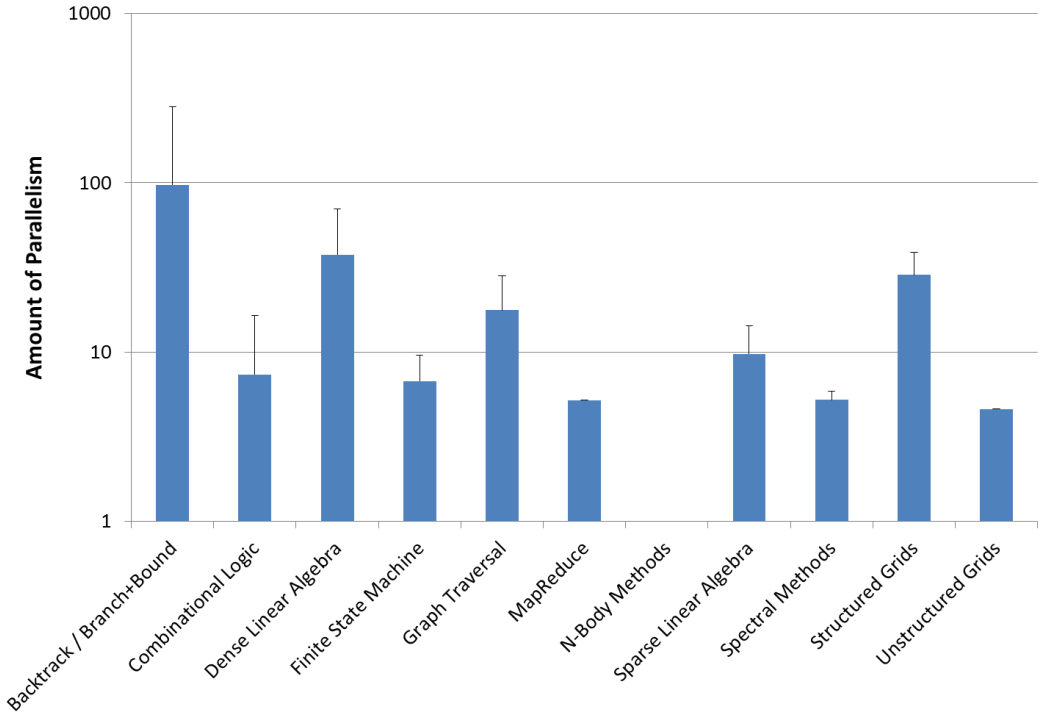


Figure 11: Task-level parallelism in Motifs

Behind these relatively large mean parallelism values is a high level of skew, where many cycles have only a small amount of parallelism, while a small number of others have

enormous parallelism. The following breakdown, again using the underlying data from Cabezas and Stanley-Marbell, illustrate this skew for the unstructured grid motif:[25]

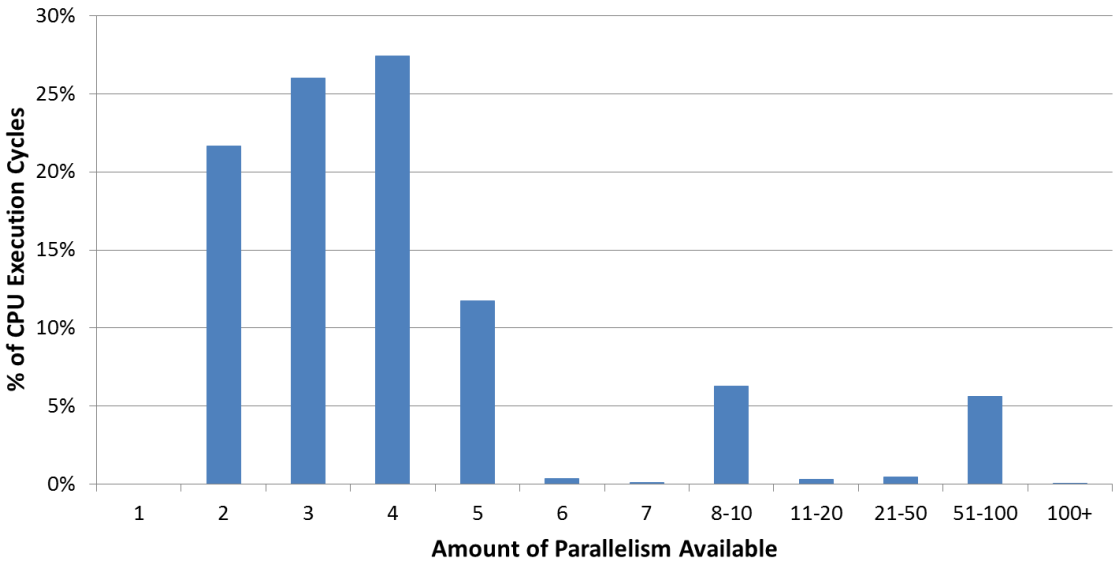


Figure 12: Task-level parallelism in the Unstructured Grids Motif

This implies that for ~85% of the program time there is task-level parallelism of 5 or less, but that for another 5% of the program, there is 51-100 task-level parallelism available. Amdahl's Law implies that such a distribution would lead to rapidly decreasing benefits from any additional cores above 5, since even making the remainder of the calculations infinitely fast would still leave 85% of the execution cycles as-is.

Using this type of analysis, one might imagine using the Cabezas and Stanley-Marbell data to deduce the time it would take for a computer with 2 cores to run these programs and to compare that with what would be implied by a 4-core machine, an 8-core machine, and so on. This approach would allow the strong scaling profile of each motif to be deduced.

Unfortunately, such an analysis fails for several reasons. First, it isn't clear that strong scaling is the correct measure. It is plausible that problem-sizes also escalate over time, implying that weak scaling could be the correct measure. Second, there may be additional sources of parallelism, for example instruction or data parallelism, which continue to improve performance. However, even if neither of these deficiencies existed, the broader goal of aggregating these data to the software level would not be possible since Cabezas and Stanley-Marbell only calculate these detailed values for roughly half of the motifs. For the others the information is coarser, so these calculations cannot be done. Thus, despite the

[25] Based on the work by Cabezas and Stanley-Marbell (2011), from a private communication.

appeal of using the Cabezas and Stanley-Marbell approach, the mapping produced by David Sheffield and Kurt Keutzer is what is used here.

Having settled on a measure of parallelism for each motif, it is important to understand the relative importance of each motif in each class of software.

The Berkeley Software Parallelism Survey elicits the motifs that respondents believe are being used in a class of software such as databases. This is done using the following questions:[26]

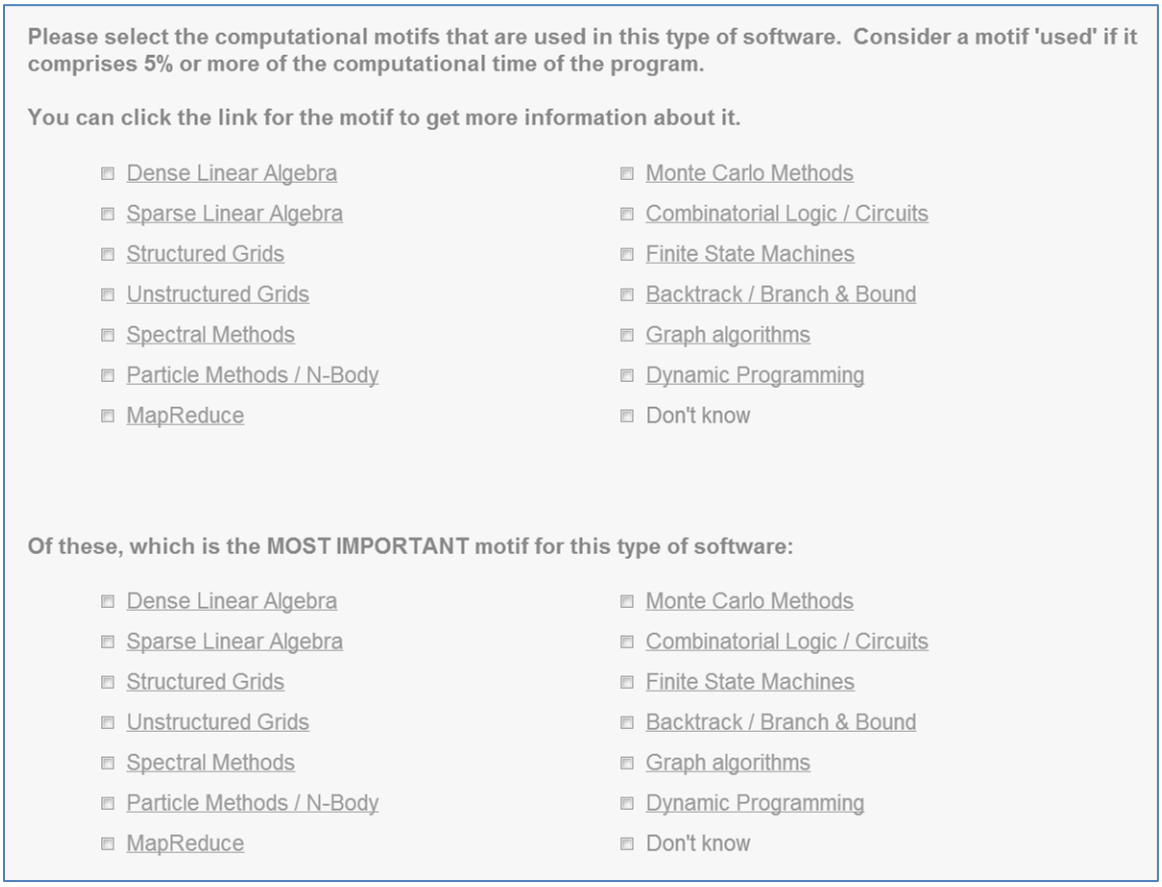


Figure 13: Motif question from Berkeley Software Parallelism Survey

Figure 13 and Figure 14 reflect the answers given by survey respondents to these questions. These are divided up into the answer for all motifs 'used', and just the most important (or 'primary') motif.[27]

[26] Notice that Monte Carlo simulation, which can be considered a special case of Map-Reduce, is here split out for simplicity.

[27] Caution: for some of the software classes, the number of respondents are low, so these have proportionally larger standard errors (not shown).

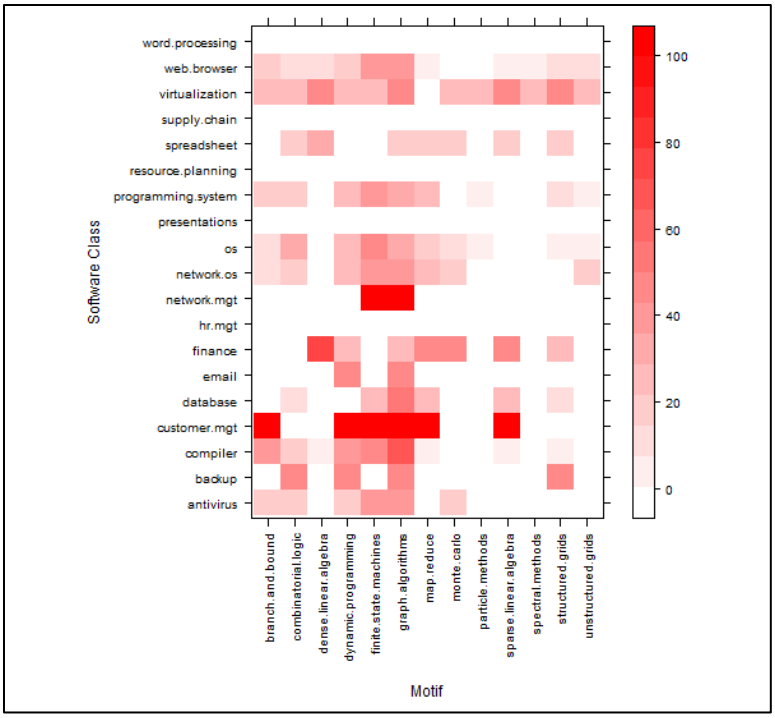


Figure 14: Survey responses for motifs used in 5%+ of software computational time (% of respondents)

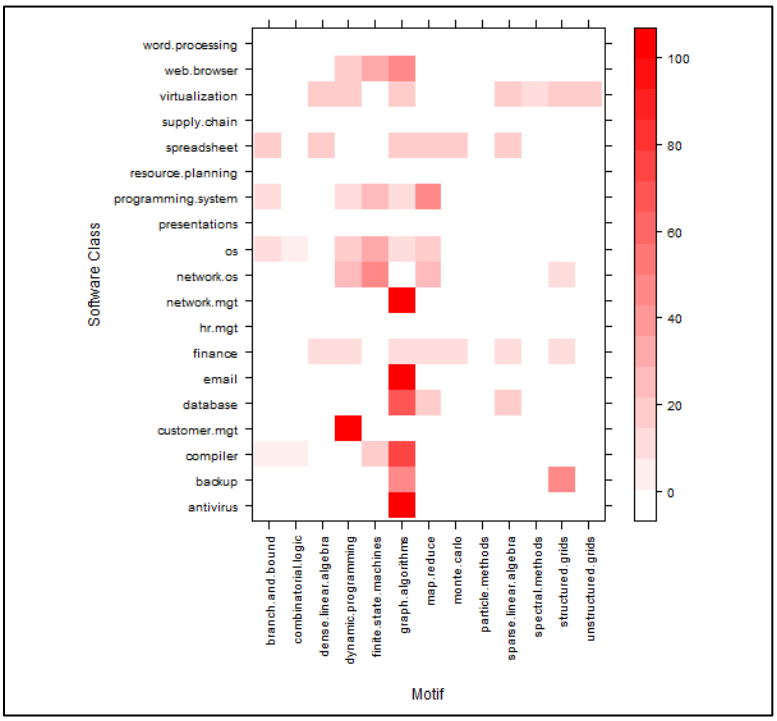


Figure 15: Survey responses for the primary motif used in software (% of responses)

Focusing on the primary motif shows, for example, that recipients believe that graph algorithms dominate the computation for antivirus software, whereas customer relationship

management software (customer.mgt) seems to be dominated by dynamic programming. For other categories, such as operating systems (os), different respondents chose different motifs as most important. If one considers this division of votes as a weighting, one can interpret this as saying that the parallelism in antivirus software corresponds to that of graph algorithms, whereas the parallelism in operating systems is a weighted average of the six motifs chosen for it.

Using these weightings, and the motif parallelism identified by Sheffield and Keutzer,[28] produces the levels of parallelism for software shown in Figure 16.

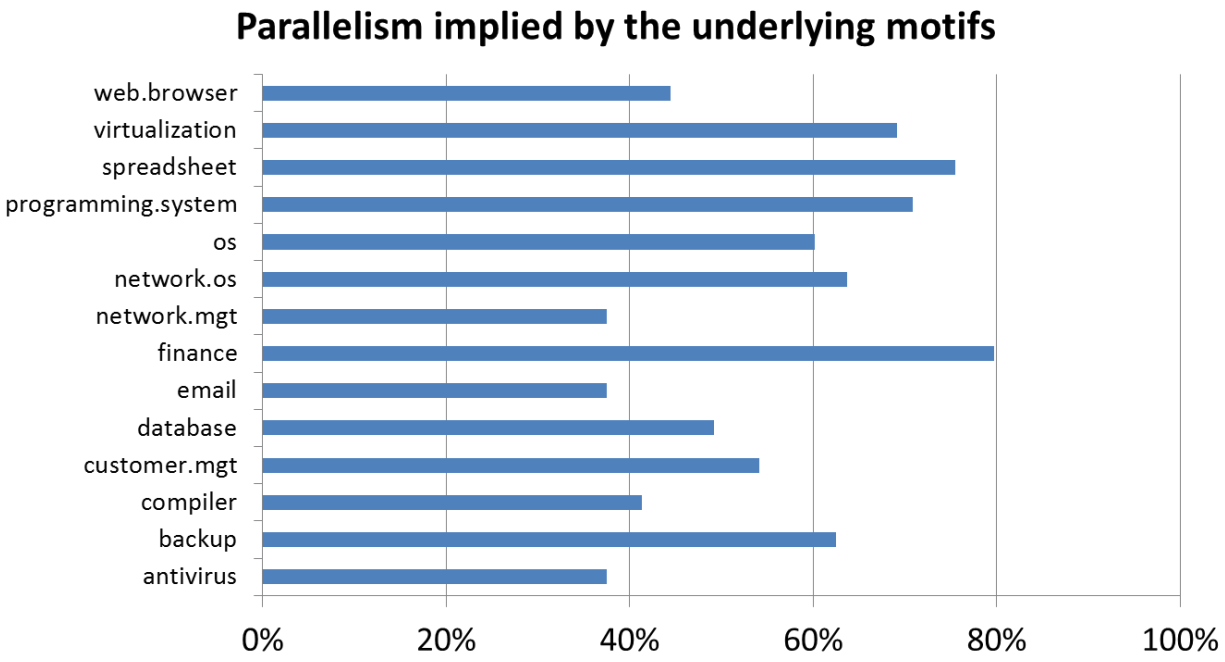


Figure 16: Software parallelism ratings implied by motif analysis

This analysis provides a second assessment of the parallelism in software. The next section outlines how these results compare to those obtained from the survey.

[28] For this chart, the author assigns the four categories from the Sheffield / Keutzer work to the ranges: 0-25%, 25%-50%, 50%-75%, 75%-100%, and then assigns a numeric value at the mid-point of those ranges. The overall parallelism for each category is assumed to the simple average of the data, task, and instruction level parallelisms. Thus, for example, Sparse linear algebra, with its assessments of green-red-green for data task and instruction level parallelism is converted into the percentages: 37.5% (midpoint 25%-50%) – 87.5% (midpoint 75%-100%) – 37.5% (midpoint 25%-50%). These are then averaged to get 54% as its overall parallelism ranking.

3.2.5 Comparing Survey- and Motif-based measures of Parallelism

Sections 3.2.3 and 3.2.4 derive two alternative methods for creating software-level measures of parallelism, one survey-based, the other motif-based. This section compares these two, testing whether they corroborate or contradict each other.

This comparison should be made using the ordinal rankings of the programs (most-to-least parallel), rather than their cardinal value, since the cardinal values are calculated from different scales (1-7 for the survey, and the average of three 1-4 values for the motifs).

The following graph presents this comparison, showing the ranking of software based on each method. The rankings are constructed such that higher values indicate more parallelism. Ties are shown at the same ranking, and the black line indicates the pattern that would be present if the two measures agreed perfectly:

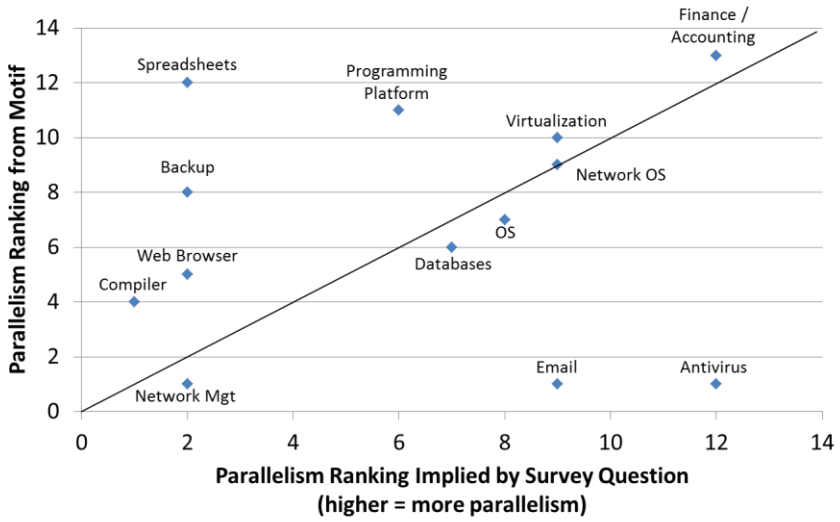


Figure 17: Comparison of Software Parallelism Ratings for Motif- and Survey-based analyses

In general, this shows only weak agreement between the two measures, with particular disagreement over software classes such as spreadsheets and antivirus software. Determining which measure is superior (and thus should be used) requires appealing to other data. For example, for spreadsheets, there are some benchmarks for particular types of tasks. These suggest that for some business applications, for example as used in stock tracking and trading, there is relatively little parallelism in spreadsheets – supporting the survey results. [29] Antivirus providers have also made strong claims about the parallelism they can use in scanning – which is also more in line with the survey results.[30] Thus, based

[29] See, for example, ExcelTrader (2011).

[30] See, for example, Kaspersky (2012).

on these findings, as well as the need for additional assumptions to generate the motif-based measures, the survey results seem to be more reliable, and are used henceforth as the main measure of software parallelism.

3.2.6 Limitations of the software parallelism measure

It is important to note that all of the measures of software parallelism outlined in Section 3.2 still have some important limitations. For the latter two measures, the survey- and motif-based measures of software parallelism, they rely on a survey conducted in 2011-2012, which post-dates the software data (2001-2007) on which it is used. All four measures also omit other factors which might be important, for example in-house customization of software or the replacement rates of software products.

3.3 Aggregating Parallelism

Having established the data and methodology for which software firms are using and for how much parallelism is in those pieces of software, this section considers how to aggregate the impact of a set of software to the firm level.

The challenge in aggregation is how to weight each piece of software. It is reasonable to assume that software provides different benefits to different firms. Moreover, the additional benefit from speed improvements would themselves vary across software programs – with additional speed being very valuable in some areas, and not at all valuable in others. As a result, properly capturing the complexity of aggregation would involve looking at a firm's usage of each piece of software and understanding how it fits (and contributes to) the production process.

Such an analysis, even for a single company, would be an enormous task. Doing it for the entire set of firms in the sample would be enormously more so. As a result, a simpler approach is chosen instead: taking the unweighted average of the software parallelism present in each of the firm's software packages. While clearly limited for all the reasons articulate above, this has the benefit of being straightforward. This is the metric used for the remainder of this thesis.

In Thompson (Working Paper) a second aggregation approach is also tested. It compares the results from using the average of the parallelism scores (as outlined above), to the results from averaging the parallelism rankings (as were used in the comparison in Section 3.2.5). Making this change has little impact on the results, implying a level of robustness to this choice.

Another approach (not yet attempted) for doing a weighted average of software programs would be to use software purchase price. This would rely on the assumption that software producers who create software that is more valuable for their clients can charge more. Because of competition, market pricing strategies, established customer bases, and the low cost of copying software, it is not obvious that this assumption should hold (as, for example, freeware demonstrates).

4 Results

This section presents the results of using the methodology outlined above to produce a firm-level measure of software parallelism. It then discusses the limitations on this measure and the steps that are being taken to improve them.

4.1 Firm Software Parallelism

Using the Harte Hanks firm software data[31] and the Berkeley Software Parallelism Survey data, it is possible to produce the parallelism scores for Swedish firms. As outlined above, this is done by taking a simple average:

$$Firm\ Parallelism_i = \frac{1}{n}\sum_{j=1}^{n} In\ Practice\ Software\ Parallelism_j$$

Here i indexes the firm[32]and j indexes the software classes being used by that firm. This produces the following distribution:

[31] This data shown here represents the sub-sample for which sufficient data was available for this analysis. In particular, the firm needed to have been surveyed in 2003 and there needed to be no missing data for the variables described in Thompson (Working Paper) in Equation 1.

[32] Technically these data are at the establishment level (i.e. a place where people are employed, for example a factory). Since many firms only have a single establishment, the term firm is used here for convenience. The software data, to which this is being aggregated, is similarly at the establishment level.

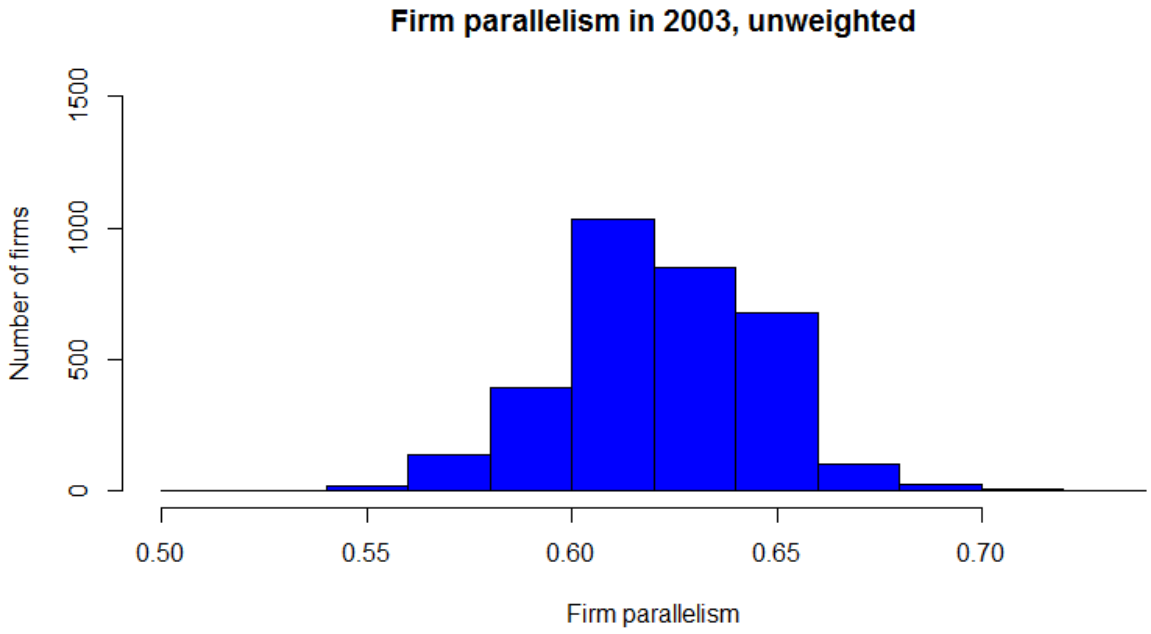


Figure 18: Firm Software Parallelism Distribution

As the graph in Figure 18 shows, the clustering of firm parallelism is relatively narrow. This is, in part, because (as was shown in Figure 6), survey results assessed the parallelism of most software in the mid-range. Another contributing factor is the distribution of software packages, where a set of applications are common across many firms (Windows, Microsoft Office, and so on), see Figure 4 for details. These contribute to each firm's parallelism score, which also narrows the results.

4.2 Limitations and Future Improvements

There are a number of important limitations to this measure. This section highlights the most important of these and discusses the next steps that are planned to improve them.

Perhaps most importantly, this measure treats each class of software (e.g. databases) as having a single level of parallelism, whereas in reality there is significant variance within software classes on how well multicore can be exploited. An example of this is the PostGreSQL database, for which the development of parallel query processing only *began* in

2009,[33] in contrast with other databases, such as IBM's DB2, which have had this functionality for much longer.[34]

As mentioned in Section 3.2.2, this analysis of within-class variance will be addressed by incorporating data on when each piece of software incorporated new features (e.g. multithreading). Much of this data was already gathered in the summer of 2012, and the remainder will hopefully be completed in 2013. This project will also help address the second major drawback from this parallelism measure: that it only incorporates single snapshot in time. For a summary of the types of data that are being gathered for this, see Appendix C.

A second approach to addressing the issues of within-class variance and changes over time would be if software / hardware providers shared their benchmarking data, as discussed in Section 3.2.1.

There is also within-program variance in parallelism arising from different inputs. So, for example, someone calculating payroll deductions using a financial accounting program would have more parallelism if calculating the accounts for 1000 people, than for 10. Benchmarking data could also help to disentangle these effects.

Another limitation of this work is the sample size and geographic scope. Considering only Swedish firms restricts the industries being considered and limits the sample to the relatively small number of firms in that country. Data is already in hand to expand the sample to Spain, France and Italy, and the author is currently being added to a U.S. Census Bureau request to get access to the data for U.S. firms. The addition of these countries may also extend the sample to beyond 2007, which would valuable for seeing the results of this effect over longer periods.

The sample size for the Berkeley Software Parallelism Survey is also a limitation for this work. It leads to some types of software being estimated with only a small number of responses. To address this, a second wave of the survey is planned. The list of recipients for this second wave is already prepared, and the second wave should go out in early 2013.

As these additional data become available, the estimates listed herein will be improved in future research.

[33] PostGreSQL Wiki (2012).

[34] Since 1995 – see Favero (2008).

5 Policy Implications

In other work related to this project, Thompson (working paper) uses the measure developed herein and combines it with data on firm productivity[35] to assess the consequences that software parallelism had on post-2005 firm performance. Productivity is chosen as the metric of firm performance since productivity increases are understood to both increase overall welfare (by yielding more output for the same inputs), and because they are an important source of national competitive advantage.

Analysis in that paper shows that firms with greater levels of software parallelism did *not* grow their productivity more rapidly than firms with low levels of software parallelism during the *clock-speed era*. That is, during the period where this difference should not have mattered, firms with greater or lesser levels of parallelism grew productivity similarly. After 2004, once multicore chips were introduced, this changed, with firms with greater levels of software parallelism growing more productivity more rapidly. Preliminary estimates suggests that the size of this effect is an increase of 0.3%-0.7% in productivity per year from 2005-2007 as a result of having one standard deviation more firm software parallelism. In economic terms, this is a large effect. If further testing and refinement continue to support this magnitude of effect, it will have important public policy implications. These might include implications for national productivity and competitiveness, as well as broad changes in which sectors are contributing productivity (and likely growth) to the economy. It will also have important ramifications for industry – both for firms producing hardware / software and those using them.

6 Conclusion

The National Research Council has highlighted the importance of the changeover to multicore architectures on software performance. They predict strong economic impacts for firms as a result of the change.

This thesis is part of a larger research project that aims to quantify those effects. The main contribution of this thesis is to build a measure of how well firms can take advantage of multicore chips, which is a key ingredient in addressing this important problem. This measure is built up using unique data on firm software usage from the marketing firm Harte Hanks, and combining it with results from the Berkeley Software Parallelism Survey – a

[35] A measure of the output that a firm can produce for a given set of inputs. See, for example, Syverson (2011) for a discussion of this.

survey of parallelism experts run for this thesis. Alternative formulations for creating this parallelism measure are also discussed and evaluated, with the reasons for choosing the main formulation clearly expressed. Finally, this thesis highlights the limitations of this measure and discusses the other on-going initiatives that are underway to address them.

This thesis also reports additional results from the Berkeley Software Parallelism Survey that may be of interest to those assessing the presence of different types of parallelism or different computational motifs in mainstream software programs.

The changeover to multicore is having, and will continue to have, important implications for firms. Although this thesis provides only a piece of the analysis needed to assess that impact, it nevertheless provides a crucial step in answering this important question.

By looking at parallelism and its underlying algorithmic / motif basis, this thesis also lays the groundwork for looking at the effect of parallelism in distributed computing environments, such as Cloud Computing, which are continuing to grow in importance.

Appendix A Database Survey Questions and Number of Respondents

Database Management Systems

e.g. Microsoft SQL Server, Oracle, IBM DB2, Sybase, Microsoft Access, Informix SQL, MySQL

In theory, how parallelizable is this type of software using the following techniques:

Click on the links to get more information about the types of parallelism.

	1. Not at all	2. Slightly	3. Partially	4. Somewhat	5. Moderately	6. Mostly	7. Completely	Don't Know / NA
Data Parallelism	○	○	○	○	○	○	○	○
Instruction Parallelism	○	○	○	○	○	○	○	○
Single-user Task Parallelism	○	○	○	○	○	○	○	○
Task Parallelism via multiple users	○	○	○	○	○	○	○	○
Overall (all types of parallelism)	○	○	○	○	○	○	○	○

If you know how parallelized the current implementations of the following software packages are, please rate them below. Please base this on overall parallelizability (all types of parallelism):

	1. Not at all	2. Slightly	3. Partially	4. Somewhat	5. Moderately	6. Mostly	7. Completely	Don't Know
Microsoft SQL Server	○	○	○	○	○	○	○	◉
Oracle	○	○	○	○	○	○	○	◉
IBM DB2	○	○	○	○	○	○	○	◉
Sybase	○	○	○	○	○	○	○	◉
Microsoft Access	○	○	○	○	○	○	○	◉
Informix SQL	○	○	○	○	○	○	○	◉
MySQL	○	○	○	○	○	○	○	◉
PostgreSQL	○	○	○	○	○	○	○	◉
Other:	○	○	○	○	○	○	○	◉

Number of respondents choosing to answer questions about each type of software:[36]

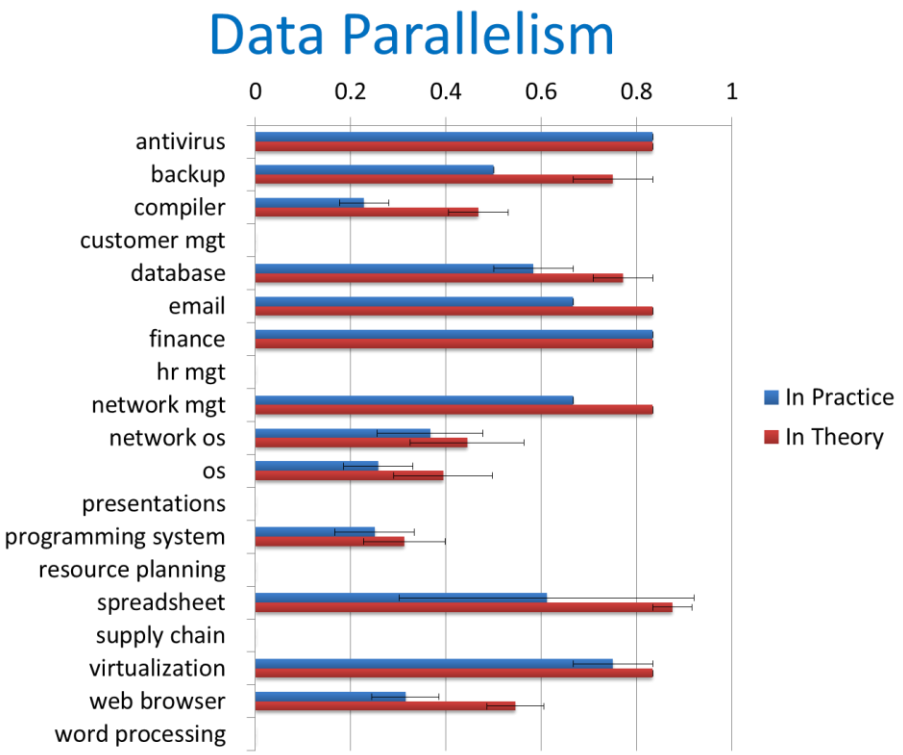
Software Class	n
Database Management Systems (e.g. SQL Server)	16
Accounting / Finance (e.g. SAP)	5
Customer Relationship Management / Salesforce automation (e.g. SAP CRM)	1
Human Resources Management (e.g. Oracle HRMS)	0
Manufacturing / Materials Resource Planning (e.g. SAP MRP)	1
Network and Systems Management (e.g. HP Openview)	2
Supply Chain Management (e.g. SAP SCM)	1
Remote Desktop / Terminal Services / Virtualization (e.g. Microsoft Terminal Server)	7
Operating Systems (e.g. Microsoft Windows)	29
Network Operating Systems (e.g. Windows Server)	14
Programming Platforms / Web Services (e.g. Microsoft .NET)	20
Anti-virus (e.g. Symantec Norton Antivirus)	6
Compilers (e.g. C++ / Java Compiler)	44
Backup (e.g. Symantec Backup)	3
Word Processing (e.g. Microsoft Word)	3
Spreadsheets (e.g. Microsoft Excel)	8
Presentation (e.g. Microsoft Powerpoint)	2
Web Browsers (e.g. Microsoft Internet Explorer)	25
Email / Scheduling (e.g. Microsoft Outlook)	2

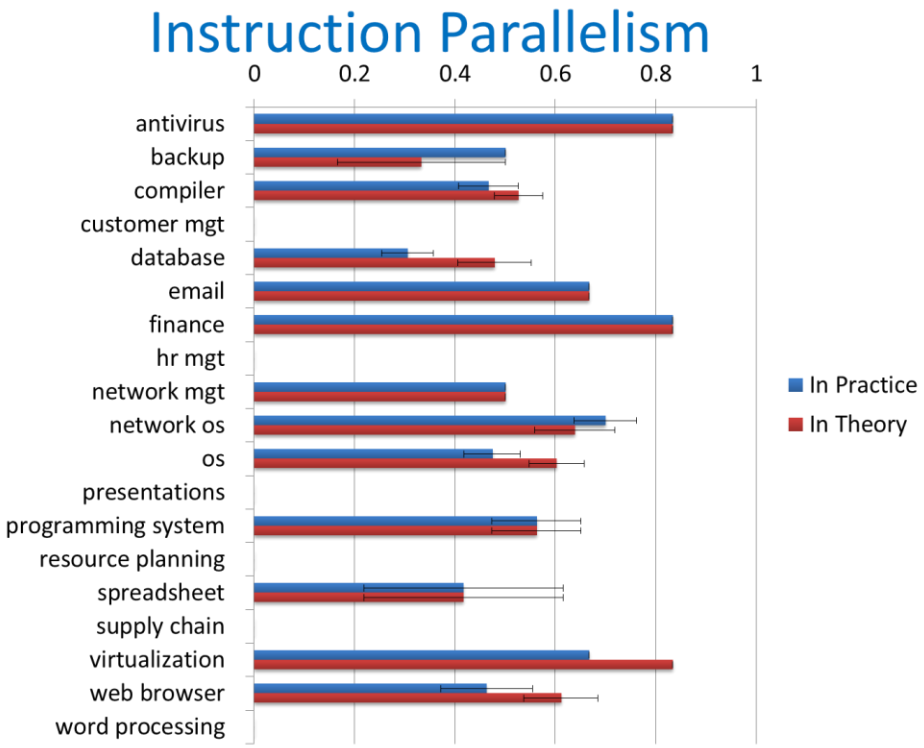
Table 5: Number of respondents to the sections of the Berkeley Software Parallelism Survey

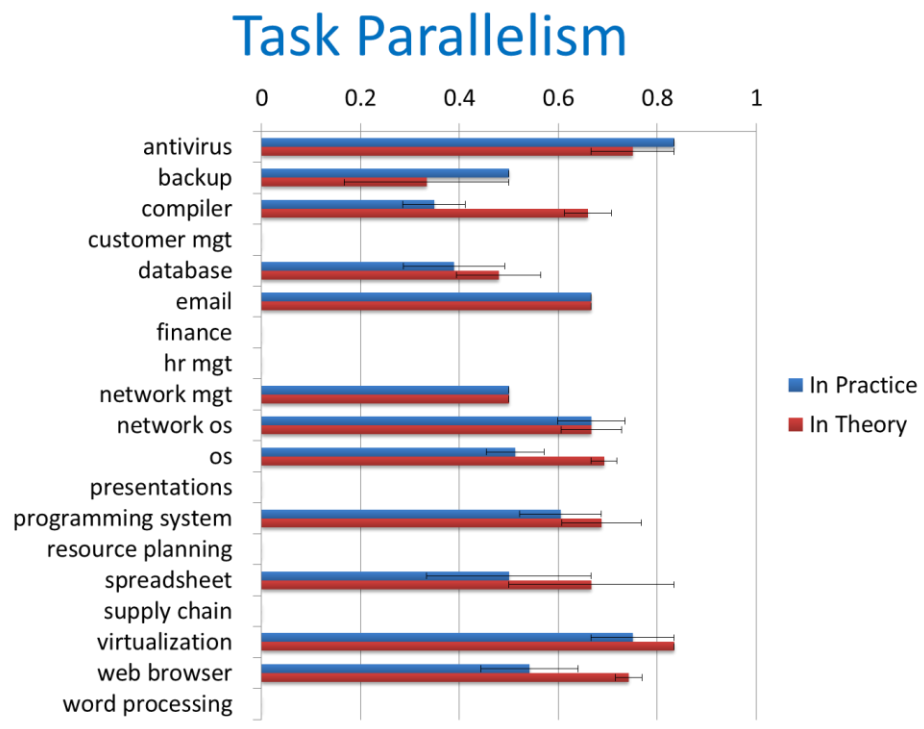
[36] Respondents may not have answered sub-questions, even after selecting into answering the broader group of questions about that class of software

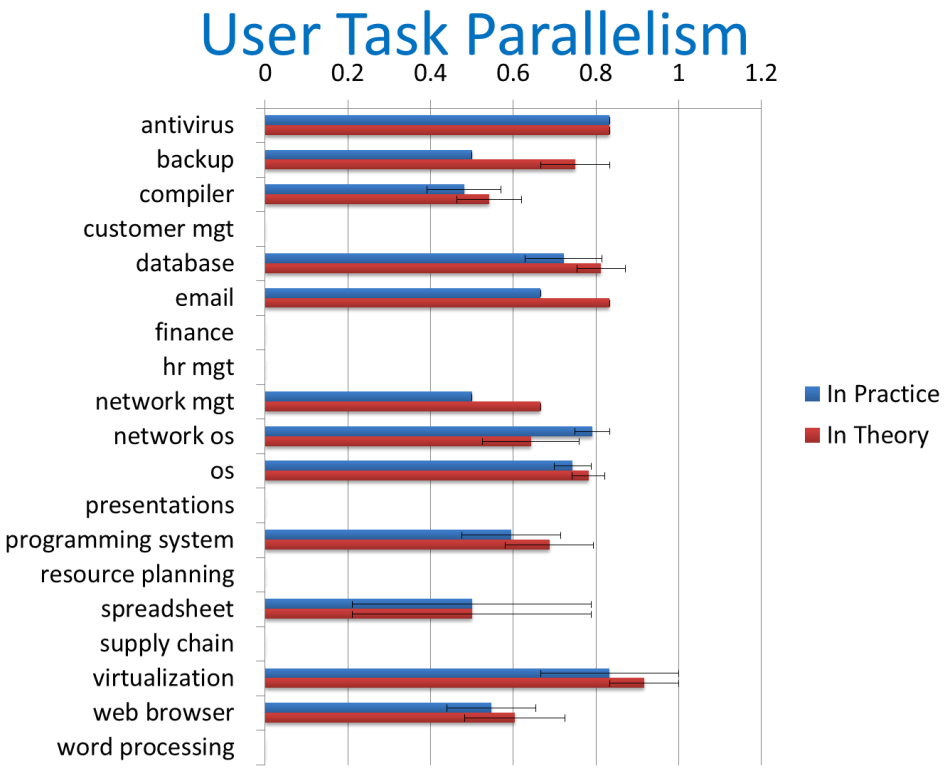
Appendix B Reports of parallelism in software types

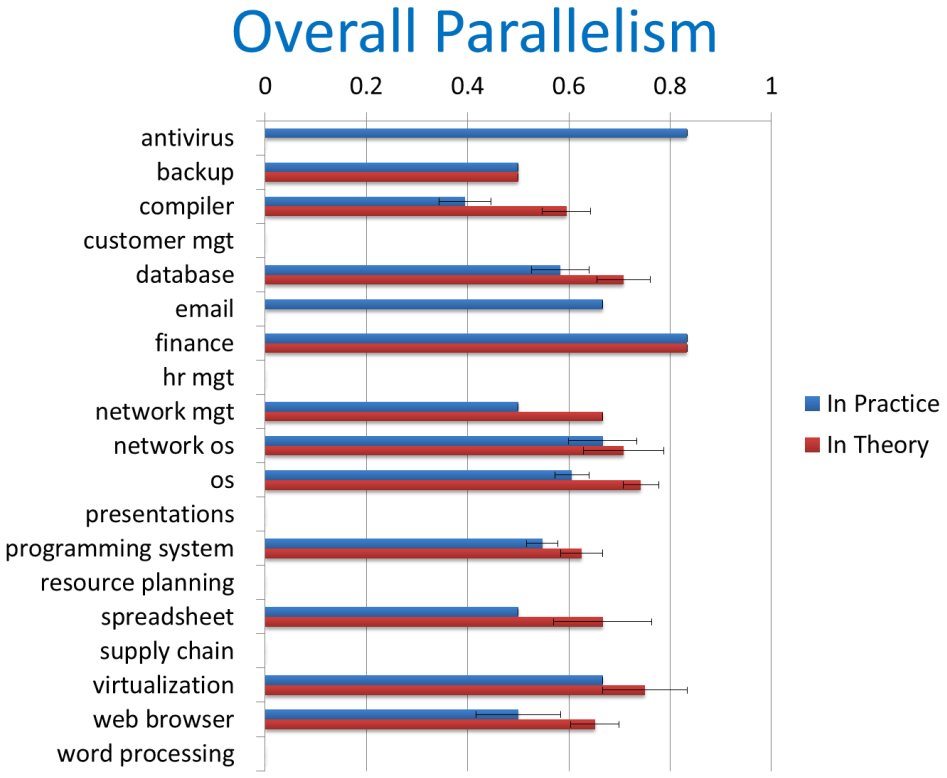
The full set of data from this survey will be posted online once subsequent survey waves are complete and their data tabulated. The data below represent some of the most important results from the first wave of the survey.



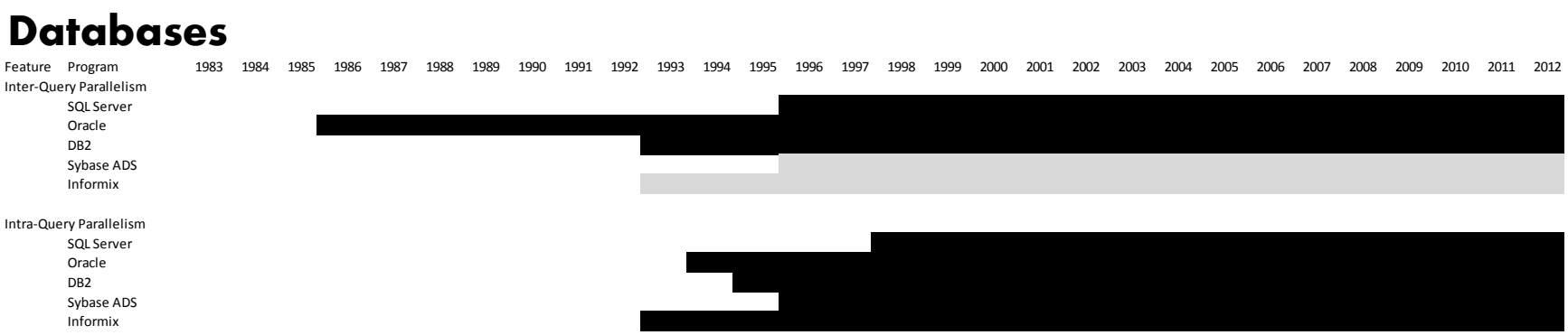

Instruction Parallelism
0
0.2
0.4
0.6
0.8
1
antivirus
backup
compiler
customer mgt
database
email
finance
hr mgt
network mgt
network os
os
presentations
programming system
resource planning
spreadsheet
supply chain
virtualization
web browser
word processing
In Practice
In Theory


Task Parallelism
0
0.2
0.4
0.6
0.8
1
antivirus
backup
compiler
customer mgt
database
email
finance
hr mgt
network mgt
network os
os
presentations
programming system
resource planning
spreadsheet
supply chain
virtualization
web browser
word processing
In Practice
In Theory


User Task Parallelism
0
0.2
0.4
0.6
0.8
1
1.2
antivirus
backup
compiler
customer mgt
database
email
finance
hr mgt
network mgt
network os
os
presentations
programming system
resource planning
spreadsheet
supply chain
virtualization
web browser
word processing
In Practice
In Theory


Overall Parallelism
0
0.2
0.4
0.6
0.8
1
antivirus
backup
compiler
customer mgt
database
email
finance
hr mgt
network mgt
network os
os
presentations
programming system
resource planning
spreadsheet
supply chain
virtualization
web browser
word processing
In Practice
In Theory

Appendix C Example of when parallelism is incorporated into software programs



Bibliography

Asanovic et al. 2006. *The Landscape of Parallel Computing Research: A View from Berkeley.* UC Berkeley Electrical Engineering and Computer Sciences, Technical Report No. UCB/EECS-2006-183.

Aral, Sinan, Erik Brynjolfsson and D.J. Wu. 2006. *Which came first, IT or productivity? The virtuous cycle of investment and use in enterprise systems.* Twenty-Seventh International Conference on Information Systems, Milwaukee 2006.

Barroso, Luiz A., Kourosh Gharachorloo, Andreas Nowatzyk and Ben Verghese. 2000. *Impact of Chip-Level Integration on Performance of OLTP Workloads.* Sixth International Symposium on High-Performance Computer Architecture (HPCA), January 2000.

Berkeley View. 2008. *The Landscape of Parallel Computing Research: A View from Berkeley.* Berkeley Parallel Computing Wiki, online at http://view.eecs.berkeley.edu/wiki/Main_Page.

Bloom, Nicholas, Raffaella Sadun and John Van Reenan. 2012. *Americans do I.T. Better: US Multinationals and the Productivity Miracle.* American Economic Review.

Brock, David. 2006. Understanding Moore's Law: Four Decades of Innovation. Chemical Heritage Foundation, Philadelphia.

Bureau Van Dijk. 2007. *Orbis.* Data extract of income statement and establishment data for Sweden for 2001-2007.

Brynjolfsson, Erik and Lorin Hitt. 2003. *Computing productivity: Firm-level evidence.* MIT Sloan Working Paper 4210-01, http://ebusiness.mit.edu/research/papers.html.

Cabezas, Victoria C., and Phillip Stanley-Marbell. 2011. *Parallelism and Data Movement Characterization of Contemporary Application Classes.* SPAA '11, June 4-6, 2011.

Cabezas, Victoria C., and Phillip Stanley-Marbell. 2011b. *Quantitative Analysis of Parallelism and Data Movement Properties Across the Berkeley Computational Motifs.* CF '11, May 3-5, 2011.

Chang, Fay et al. 2006. *Big Table: A Distributed Storage System for Structured Data.* OSDI: Seventh Symposium on Operating System Design and Implemenation.

Council on Competitiveness. 2005. *High Performance Computing and Competitiveness.*

Dean, Jeffrey and Sanjay Ghemawat. 2004. *MapReduce: Simplified Data Processing on Large Clusters.* OSDI 2004.

De Gelas, Johan. 2005. *The Quest for More Processing Power, Part One: "Is the single core CPU doomed?"* online at http://www.anandtech.com/show/1611.

ExcelTrader. 2011. *Excel Benchmark 2011: An Excel Speed Test (with trading functions).* ExcelTrader online at http://exceltrader.net/984/benchmark_et-xls-an-excel-benchmark-for-traders/.

Favero, Willie. 2008. *DB2 History 101: Version 4.* Toolbox database blogs, online at http://it.toolbox.com/blogs/db2zos/db2-history-101-version-4-23970.

Frost and Sullivan. 2005. *World Market for Dual Core Processors.* Market research report.

Gordon, Robert J. 2000. *Does the "New Economy" measure up to the great inventions of the past?* NBER working paper 7833.

Gower, Anabelle and Michael Cusumano. 2002. Platform Leadership. Harvard Business Review Press.

Harte Hanks. 2002. *CI Technology Database Methodology.* Harte Hanks marketing distribution document.

Harte Hanks. 2006. *Relational Database Format: A guide to using the CI Technology Database with relational database software.* Online at: http://www.europe.hartehanksmi.com/support/downloads/RDF_Manual.pdf.

Harte Hanks. 2007. *CI Technology Database.* Data extract from Sweden, 2001-2007, including establishment and equipment data.

Hennesy, John and David Patterson. 2007. Computer Architecture: A Quantitative Approach. Morgan Kaufmann Publications, San Francisco.

Hillis, W. Daniel and Guy L. Steele, Jr. 1986. *Data Parallel Algorithms.* Communications of the ACM, Vol. 29, No. 12, December 1986.

IDC. 2002. *Worldwide IT Spending by Vertical Market Forecast, 2001-2006.* International Data Corporation.

Intel. 2000. *Intel Introduces the Pentium 4 Processor.* Intel News Release, online at: http://web.archive.org/web/20070403032914/http://www.intel.com/pressroom/archive/releases/dp112000.htm.

Jorgenson, Dale, Kevin Stiroh, Robert Gordon and Daniel Sichel. 2000. *Raising the Speed Limit: U.S. Economic Growth in the Information Age.* Brookings Papers on Economic Activity, Vol. 2000, No 1 (2000), pp. 125-235, Brookings Institution Press.

Jorgenson, Dale, Mun Ho, and Jon Samuels. 2010. *Information Technology and U.S. Productivity Growth: Evidence from a Prototype Industry Production Account.* Prepared for *Industrial Productivity in Europe: Growth and Crisis*, online at: http://www.economics.harvard.edu/faculty/jorgenson/files/02_jorgenson_ho_samuels%2B19nov20101_2.pdf.

Kaspersky. 2012. *Kaspersky Anti-virus for Novell NetWare.* Kaspersky Lab, online at http://www.kaspersky.com/anti-virus_novell_netware.

Longbottom, Roy. 2012. *Roy Longbottom's PC Benchmark Collection.* Online at http://www.roylongbottom.org.uk/.

Malone, Michael. 1995. The Microprocessor: A Biography. Springer-Verlag, New York.

National Research Council. 2005. Getting up to Speed: The Future of Supercomputing. National Academies Press, Washington, D.C.

National Research Council. 2011. The Future of Computing Performance: Game Over or Next Level? National Academies Press, Washington, D.C.

Olukotun, Kunle, Lance Hammond, Herb Sutter, Burton Smith, Chris Batten, Krste Asanoviç and Angelina Lee. 2012. CPU DB. Data available at http://cpudb.stanford.edu/. Data received in 2012.

OPL Working Group. 2012. *A Pattern Language for Parallel Programming ver2.0.* ParLab Patterns Wikipage, online at: http://parlab.eecs.berkeley.edu/wiki/patterns/patterns.

PC & Tech Authority. 2009. *The greatest tech U-turns of all time: Intel and Netburst.* Online at: http://www.pcauthority.com.au/News/163122,the-greatest-tech-u-turns-of-all-time-intel-and-netburst.aspx.

PostgreSQL wiki. 2012. *Parallel Query Execution.* PostGreSQL wiki online at: http://wiki.postgresql.org/wiki/Parallel_Query_Execution.

Regan, Keith. 2004. *Intel Puts 4 GHz Processor on Back Burner.* E-Commerce Times, Oct 15, 2004. Online at http://www.ecommercetimes.com/story/37360.html.

SPEC. 2002. *SPEC's Benchmarks and Published Results.* Online at: http://www.spec.org/benchmarks.html#cpu, accessed Summer 2012.

Stata. 2012. STATA MP homepage. Online at: http://www.stata.com/statamp/, accessed Aug 2012.

Sutter, Herb. 2005. *The Free Lunch Is Over: A Fundamental Turn Toward Concurrency in Software*, online at http://www.gotw.ca/publications/concurrency-ddj.htm.

Symantec. 2005. *Enabling Multithreaded Scans.* Symantec Knowledge Base Article. Online at http://www.symantec.com/business/support/index?page=content&id=TECH101387.

Syverson, Chad. 2011. *What determines productivity?* Journal of Economic Literature, 49:2, pp. 326-365.

Thompson, Neil. Working Paper. *The Statistics of how Moore's Law has impacted firm productivity.* U.C. Berkeley Statistics Master's Thesis. Results as of Fall 2012.

Watt, Martin, Lawrence D. Cutler, Alex Powell, Brendan Duncan, Michael Hutchinson, and Kevin Ochs. 2012. *LibEE: A Multithreaded Dependency Graph for Character Animation.* DigiPro '12, the Proceedings of the Digital Production Symposium, Pp 59-66.