

# The Landscape of Parallel Computing Research: A View from Berkeley

Krste Asanović, Rastislav Bodik, Bryan Catanzaro, Joseph Gebis, Parry Husbands, Kurt  
Keutzer, David Patterson, William Plishker, John Shalf, Samuel Williams, and  
Katherine Yelick

December 18, 2006

## 并行计算研究前景：伯克利视角 (并行计算与 13 个小矮人)

翻译指导、修订及评论：龚春叶\*、刘杰、甘新标、李胜国、杨博

国防科技大学计算机学院

2018.05.01

2018 年春季《面向高性能计算的性能评测与优化技术》研究生课程作业

[\\*gongchunye@nudt.edu.cn](mailto:*gongchunye@nudt.edu.cn)，注：对文章的简单评论用【\*\*\*】表示。

# The Landscape of Parallel Computing Research: A View from Berkeley



*Krste Asanovic  
Ras Bodik  
Bryan Christopher Catanzaro  
Joseph James Gebis  
Parry Husbands  
Kurt Keutzer  
David A. Patterson  
William Lester Plishker  
John Shalf  
Samuel Webb Williams  
Katherine A. Yelick*

Electrical Engineering and Computer Sciences  
University of California at Berkeley

Technical Report No. UCB/EECS-2006-183  
<http://www.eecs.berkeley.edu/Pubs/TechRpts/2006/EECS-2006-183.html>

December 18, 2006

Copyright © 2006, 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

We'd like to thank the following who participated in at least some of these meetings: Jim Demmel, Jike Chong, Armando Fox, Joe Hellerstein, Mike Jordan, Dan Klein, Bill Kramer, Rose Liu, Lenny Oliner, Heidi Pan, and John Wawrzynek.

学生列表：

负责内容	姓名	人数
Abstract+合稿	万仟、邓海莲	2
<b>翻译小组</b>		
1.0 Introduction	郑小红	1
2.0 Motivation	杜林林	1
3.0 Applications and Dwarfs	王贺春、李逸聪、张超 张朝栋、刘盼雨、王浩	6
4.0 Hardware	唐丽园、李欣瑶、张兴雷 朱摘贤、何旺宇、唐安遥	6
5.0 Programming Models	番丝江、丁凡	2
6.0 Systems Software	贲宗承、王勇	2
7.0 Metrics for Success	易亮	1
8.0 Conclusion	钟造成	1
<b>校对小组</b>		
1.0 Introduction	李雨蓉	1
2.0 Motivation	刘汝霞	1
3.0 Applications and Dwarfs	郑文旭、沈亮、潘晓东	3
4.0 Hardware	付强、谢浩程、王仁敏	3
5.0 Programming Models	钱骥	1
6.0 Systems Software	冷灿	1
7.0 Metrics for Success	怀智博	1
8.0 Conclusion	耿铭阳	1

## 摘要

近年改用并行微处理器是计算历史上的一个里程碑。业界已经为多核设计制定了通过二进制兼容性和缓存一致性来保留过去的编程范例的路线图。传统的观点认为当下每一代硅芯片的核数是上一代的两倍。

一个由伯克利研究人员组成的多学科小组近两年就这一变化进行了讨论。我们的观点是：这种软硬件并行的进阶方法可以工作在 2 个或 8 个处理器系统下，但随着 16 个和 32 个处理器系统的实现，它们可能会面临收益递减的问题，正如伴随着更高的指令级并行其效果会下降一样。

我们相信，通过研究计算频谱极端情况下的并行性的成功，即嵌入式计算和高性能计算，可以学到很多东西。这导致我们用七个问题来描述并行前景，并归纳为以下内容：

- 总体目标应是使编写在高性能并行计算系统上高效执行的程序变得容易。
- 目标应是每个芯片有 1000 个内核，这些芯片都由最高效的处理单元组成，即每个处理单元在单位耗电量、单位硅面积和单位成本下的 MIPS 为最高。
- 使用 13 个“小矮人”（dwarf）来设计和评估并行编程模型和架构，而不是传统的基准测试。（矮人是一种描述计算和通信模式的算法或者方法。）【前传叫《白雪公主与七个小矮人》。】
- “自动调优”（自调优）在翻译并行程序时应该比传统编译器发挥更大的作用。
- 为了最大限度地提高程序员的工作效率，未来的编程模型必须更加注重以人为本而非传统模式中的以硬件或应用程序为中心。
- 要想达到成功，编程模型应该与处理器的数量不相关联。
- 为了最大限度提高应用程序的效率，编程模型应支持宽泛的数据类型和成功的并行模型：任务级并行性，字级并行性和位级并行性。
- 如果程序员不能精确地通过性能计数器和能量计数器来衡量它们的影响，架构师就不应该包含对性能和能耗有显著影响的特性。

- 传统的操作系统将被解构，操作系统功能将使用库和虚拟机进行编排。
- 为了更快速地探索设计空间，可使用基于现场可编程门阵列（FPGA）的系统仿真器，该系列仿真器具有高度可扩展性和低成本特性。

由于现实世界的应用程序自然是并行的，硬件自然是并行的，所以我们需要的是仅仅是一种编程模型、系统软件和支持自然并行的体系结构。研究人员只有通过简化高度并行系统的高效编程，才有机会重构这些计算基础。

## 目录

The Landscape of Parallel Computing Research: A View from Berkeley.....	a
并行计算研究前景：伯克利视角 .....	a
(并行计算与 13 个小矮人).....	a
摘要 .....	1
1.0 引言 .....	5
2.0 动机 .....	8
3.0 应用程序和小矮人 .....	11
3.1 七个小矮人 .....	11
3.2 寻找更多的小矮人 .....	12
3.3 小矮人的组成部分 .....	19
3.4 英特尔的研究 .....	20
3.5 13 个小矮人 .....	21
4.0 硬件 .....	28
4.1 处理器：小而美 .....	28
4.2 解除访存限制 .....	33
4.3 互联网络 .....	34
4.4 通信原语 .....	36
4.5 可靠性 .....	38
4.6 性能和能耗计数器 .....	39
5.0 编程模型 .....	41
5.1 心理学研究启发下的编程模型尝试 .....	43
5.2 型号应与处理器的数量无关 .....	45
5.3 模型必须支持各种各样的数据大小和类型 .....	45
5.4 模型必须支持已有的并行性风格 .....	46
6.0 系统软件 .....	48
6.1 自动调优与传统编译器的比较 .....	48
6.2 解构操作系统的支持 .....	51
7.0 成功的衡量标准 .....	54

7.1 最大化程序员生产力【开发效率】 .....	54
7.2 最大化应用程序性能 .....	55
7.3 RAMP: 多处理器的研究加速器 .....	56
致谢 .....	61
附加评论: .....	62
参考文献 .....	64



## 1.0 引言

2005 年，当 Intel 紧随 IBM Power 4 和 Sun Microsystems 的 Niagara 的引导，随后宣布自己的高性能微处理器今后将依赖多处理器或多核时，计算行业发生了改变。新的行业术语“多核”掌握每个管芯标准内核的数量与每一个半导体工艺发生起始具有单个处理器倍增的计划。多核明显地有助于减轻多道程序的工作负载，工作负载混合了很多独立序列任务，但是单独的任务如何变得更快呢？不需要通过大幅度提高功耗性能来增强更明显的效果，从序列到中等并行计算的切换就将使编程变得更加困难。因此，多核不可能是理想的答案。

在 2005 年 2 月和 2006 年 12 月期间，一群加利福尼亚大学伯克利分校的研究员们相聚在一起从多个角度讨论了并行性。他们来自许多不同的背景，有电路设计、计算机体系结构、大规模并行计算、计算机辅助设计、嵌入式硬件与软件、编程语言、编译器、科学规划以及数值分析。我们从不同的学科中借鉴了一些关于并行性的好想法，从而形成了这份报告。我们的结论是，通过多核解决方案解决并行问题可能会失败，我们迫切需要一种新的并行硬件和软件的解决方案。

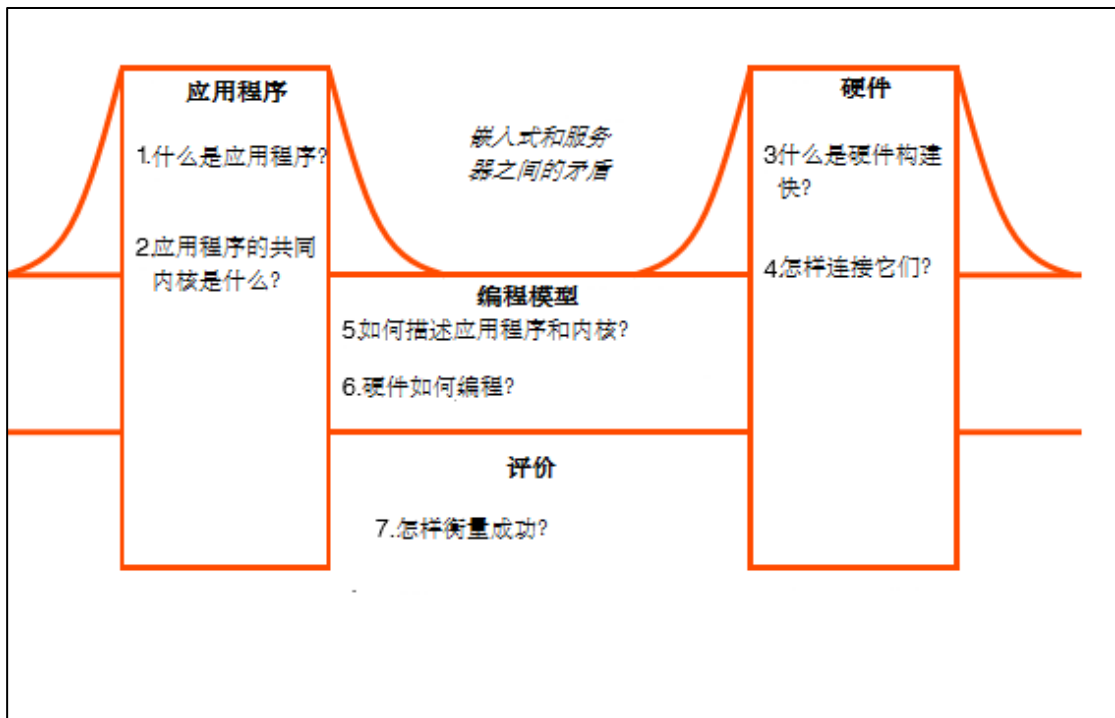


图 1 .Berkeley 观点：21 世纪并行计算的七个关键问题

【上面这个桥感觉画得特别好】

虽然旧的二进制程序和 c 程序的兼容性对产业是有价值的，而且一些研究者正试图帮助多核产品计划成功，但是我们也一直在思考更大胆的想法。我们的目标是在一个芯片上为新应用实现成百上千的处理器，我们欢迎能简化高度并行系统编程的新编程模型和新架构。我们的重点是众核“**manycore**”，而不是多核。成功的多核架构和支撑软件技术可能会在未来 30 年重置微处理器硬件和软件的路线图。

图 1 显示了我们用来构建并行计算研究领域的七个关键问题。我们并没有声称在这篇报告中有答案，但我们确实对其中的一些问题提出了非传统的具有挑衅的观点，并在别的方面提出了一些显而易见但有时会被忽略的观点。

需要注意的是，关于嵌入式和高性能计算之间的矛盾，在我们的多次讨论中都有涉及。我们认为，计算光谱在这两方面较过去更具有前瞻性：

- 首先，两者都关注能耗，无论是手机的电池寿命还是数据中心的电力和冷却费用。
- 其次，两者都会涉及硬件的使用。嵌入式系统对成本总是很敏感，但当你为高端服务器花费 1000 万到 1 亿美元时，还是需要硬件的有效使用。
- 第三，随着嵌入式软件的规模不断增加，手动调优的部分将受到限制，因此软件重用的重要性必须增加。
- 第四，现在嵌入式和高端服务器都需要连接网络，二者都需要防止不必要的接入和病毒。

因此，对于某种形式的操作系统来说，对嵌入式系统中的保护以及资源共享和调度的需求日益增加。

这两个目标之间最大的区别在于传统的嵌入式强调实时计算，计算机和程序只需要足够快才能满足最后期限，但是运行速度并不是越快越好。而在服务器计算中运行速度快通常是有价值的。随着服务器应用越来越面向媒体，实时性对服务器计算也会更加重要。本报告借鉴了嵌入式和高性能计算中许多的思想。

本报告的组织结构根据图一的七个问题进行展开。第 2 节通过提供一些指导原则来说明为什么转换到并行计算。第 3 节回顾了图一的左边部分，它代表了新的并行应用程序，并描述了最初的“七个小矮人”，这将是未来许多应用核程序的计算内核。第 4 节回顾了图一的右边部分，它是并行性的硬件，我们将分别讨

论处理器、内存、交换机的传统分类。第 5 节涵盖了编程模型，第 6 节包括系统软件，它是连接图一左右两部分的桥梁。第 7 节讨论成功的措施，并描述了一种探索并行计算的新硬件工具。最后，对我们的观点进行总结，鉴于我们在报告中讨论的主题较广泛，我们为有兴趣继续学习的读者提供了 134 份参考资料（参考文献）。

此外，我们还创建网站和博客，用来继续讨论报告中提出的观点。详情参考 [view.eecs.berkeley.edu](http://view.eecs.berkeley.edu)。

## 2.0 动机

在过去至少三十年，并行的前景一直吸引着研究者们。曾经，并行计算的研究有了初步前景并且集中了投入，但最后还是单处理器计算占了上风。然而，我们主张现在通用计算正在不可逆转地向并行体系结构发展。那么这次的转变有什么不同呢？这次朝着并行化的转变，不同于以往基于新型软件和并行化结构突破的转变，而是从传统单处理器结构上硅的使用这一更大挑战上的一次撤退。

接下来，我们抓住了一些指导原则，来精确阐述计算中的一切是怎么改变的。仿照《新闻周刊》的格式，将过时的传统概念和相应的新替换概念成对地列出。用 CW 来表示这些成对的概念。

1. Old CW: 能耗是免费的，但是晶体管很昂贵。
  - New CW 是“能耗墙”：电能是昂贵的，但是晶体管是“免费”的。也就是说我们在一个芯片上放置的晶体管数量超过了我们的电能发动的量。
2. Old CW: 如果你关心耗电量，那么你只需要关心动态耗电量。
  - New CW: 对于台式机和服务器的来说由于泄露造成的静态耗电量可以占到总耗电量的 40%。（见章节 4.1）
3. Old CW: 单片处理器的硅内部性质可靠，错误只会在引脚处发生。
  - New CW: 随着芯片的特征尺寸下降到 65nm 以下，芯片的软性和硬性错误比率都变得很高 [Borkar 2005] [Mukherjee et al 2005] 。
4. Old CW: 得益于前人的成功，我们可以继续提升抽象的等级并因此提升硬件设计的尺寸。
  - New CW: 导线延迟、噪音、交叉耦合（电容性的和电感性的）、制造变异性、可靠性（见上文）、时钟抖动、设计验证等使得 65 纳米或更小尺寸设计的开发时间和成本更高。
5. Old CW: 通过制造芯片，研究者们贡献新的结构理念。
  - New CW: 65 纳米特征尺寸的掩膜成本，用电子计算机辅助设计软件来设计这样的芯片的成本，以及设计 GHz 时钟频率的成本，意味着研究人员不能再构建可信的原型。因此，必须开发另一种评估体系结构的方法。

（见章节 7.3）

6. Old CW: 性能改进会产生较低的延迟和较高的带宽。
  - New CW: 在许多技术中, 带宽的提升至少是延迟改善的两倍。[Patterson 2004]
7. Old CW: 乘法很慢, 但加载和存储速度很快。
  - New CW 是“存储墙” [Wulf and McKee 1995]: 加载和存储缓慢, 但乘速度很快。现代微处理器进行访问动态随机存取存储器 (DRAM) 时可以用 200 个时钟周期, 但即使是浮点乘法可能只需要四个时钟周期。
8. Old CW: 我们可以通过编译器和体系结构创新揭示更多的指令级并行性 (ILP)。比如说过去的分支预测, 无序执行, 推测和超长指令字系统。
  - New CW 是“ILP 墙”(指令墙?): 寻找更多 ILP 的收益递减。[Hennessy and Patterson 2007]
9. Old CW: 单处理器性能每 18 个月翻一番。
  - New CW 是能耗墙 + 存储墙 + ILP 墙 = 障碍。图 2 绘制了近 30 年的处理器性能。在 1986 年到 2002 年之间性能是每 18 个月翻一番, 在 2006 年, 性能的发展已经低于之前的速度了。单处理器性能翻倍现在可能需要 5 年时间。
10. Old CW: 不需要花费精力去并行你的应用程序, 你只需稍微等一下就可以在一台速度快的多的串行机上运行。
  - 新的 CW: 等待更快的串行机需要很长时间 (见上文)。
11. Old CW: 增加时钟频率是改进处理器性能的主要方式。
  - New CW: 增加并行性是提升处理器性能的主要方式。(见章节 4.1)
12. Old CW: 对于多处理器程序来说, 低于线形比例的速度是失败的。
  - New CW: 鉴于切换到并行计算, 通过并行的任何加速都是成功的。

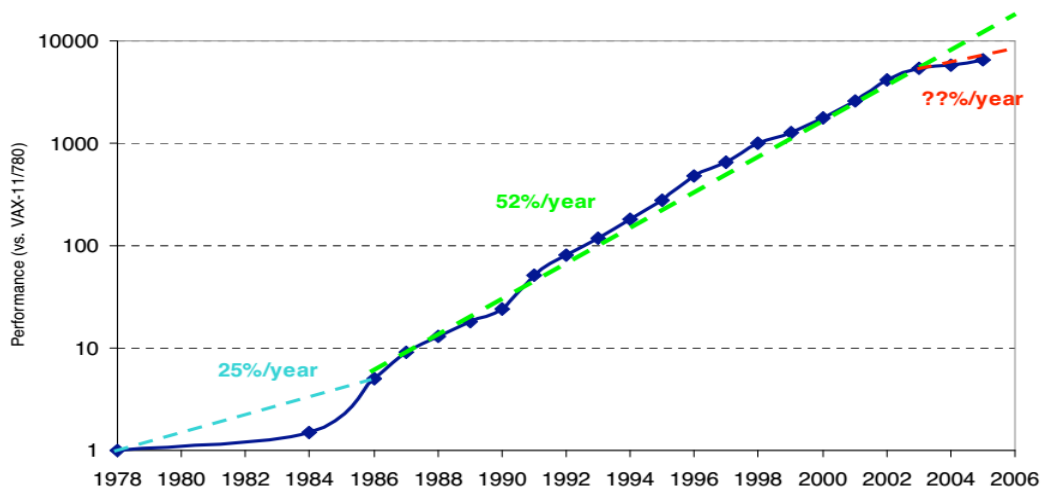


图 2. 使用整型 SPEC 程序[SPEC 2006]测量在 1978 年到 2006 年间的处理器性能提升程式。1986 年至 2002 年期间，RISC 每年帮助提高 52% 的性能，比 VAX 小型机在 1978 年至 1986 年间的提升速度快很多。自 2002 年以来，每年性能提升不到 20%。到 2006 年，如果仍以每年 52% 的速度持续发展，处理器将是三者中发展最慢的。这张图是自于[Hennessy and Patterson 2007]中的图

### 1.1.

虽然上面的 CW 对描绘了关于硬件状态的负面情况，但在也有补偿性的积极因素。首先，摩尔定律持续有效，所以我们很快就会成为现实能够将数以千计的简单处理器放在一个经济实惠的芯片上（见章节 4.1.2）。例如，思科正推出一款拥有 188 个精简指令集的计算机（RISC）核，在一个 130 纳米的单芯片上 [Eatherton 2005]。第二，芯片中这些处理器之间的通信可以具有非常低的延迟，非常高的带宽。这些单片多核微处理器展现了与传统的多芯片多处理器非常不同的设计，并预示着开发新的体系结构和编程模型的前景。第三，软件开源运动意味着软件栈可以比过去演变得更快。比如说，请注意 Ruby on Rails 的广泛使用。2005 年 12 月发布版本 1.0。

### 3.0 应用程序和小矮人

图 1 的左塔是应用程序。除了传统的桌面，服务器，科学和嵌入式应用程序之外，消费产品的重要性也在不断增加【确实如此，如移动互联网应用、金融类产品、短视频产品，但是真正对科技有大发展还是传统的科学与工程计算应用】。

我们决定提升高性能计算社区的并行体验，以了解是否有机会有机会可以让我们从更广泛的角度学习并行计算。这个假设并不是说传统的科学计算是并行计算的未来；而是在编译程序中构建的知识体系，这些程序在大型并行计算机上运行良好，且可能会在未来的并行应用程序中发挥作用。此外，许多来自其他领域的作者（例如嵌入式计算）都对其领域中未来的应用程序如何与科学计算中的问题密切相关感到惊讶。

指导和评估体系结构创新的传统方法是研究基于现有程序的基准，如 EEMBC（嵌入式微处理器基准联盟）或 SPEC（标准性能评估公司）或 SPLASH（斯坦福共享内存的并行应用）[EEMBC 2006] [SPEC 2006] [Singh 等 1992] [Woo 等 1992]。并行计算创新的最大障碍之一是目前还不清楚如何最好地表达并行计算。因此，让一组现有的源代码驱动对并行计算的调查似乎是不明智的。对于并行应用程序需求的推理，我们需要找到更高层次的抽象。【没有提到 HPL、HPCG，应用级别的计算更加重要】

我们的目标是以某种方式描述应用程序需求，而不是针对个别应用程序或用于特定硬件平台的优化，以便我们可以得到关于硬件需求更广泛的结论。我们的方法，如下所述，是定义一些“小矮人”，它们分别捕捉一类重要应用程序所通用的计算和通信模式。

#### 3.1 七个小矮人

我们受到了 Phil Colella 的启发，他确定了七种数值方法，他相信至少在未来十年内，这七种方法对于科学和工程学都是重要的[Colella 2004]。图 3 介绍了这七种小矮人，它们构成的类中，成员间关系是由计算和数据运动的相似性定义的。小矮人是在高度抽象的层次上指定的，以允许在广泛的应用范围内对他们的行为进行推理。作为特定类别的成员的程序，可以以不同的方式实现，并且底层的数

值方法可能会随着时间而改变，但需要声明的是，这些底层的方法已经延续了几代人的变化，并且在未来仍然很重要。

在以这些等价类为基础构建的数值库中，可以找到这种“等价类”存在的一些证据：例如，用于频谱分析的 FFTW [Frigo 和 Johnson 1998]，用于稠密线性代数的 LAPACK / ScaLAPACK [Blackford 等, 1996]，用于稀疏线性代数的 OSKI [Vuduc 等, 2006] [【还有 Petsc】](#)。在图 3 [【论文中图表都视为图】](#) 中我们列出了这些结构，以及专为特定小矮人创建的计算机体系结构：例如，用于 N-body 方法的 GRAPE [Tokyo 2006]，线性代数的矢量架构[Russell 1976]和 FFT 加速器[ Zarlink 2006]。图 3 还显示了在并行机器上运行时，一个小矮人成员表现出的处理器间通信模式[Vetter 和 McCracken 2001] [Vetter 和 Yoo 2002] [Vetter 和 Meuller 2002] [Kamil 等 2005]。通信模式与每个处理器本地采取的内存访问模式密切相关。

## 3.2 寻找更多的小矮人

小矮人提出了一种方法来捕获应用程序类的通用需求，同时合理地脱离了独立实现。虽然小矮人的命名来自 Phil Colella 对科学计算应用的讨论，但我们有兴趣将小矮人应用于更广泛的计算方法。这就引出了以下问题：

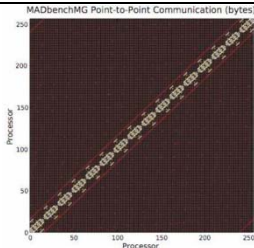
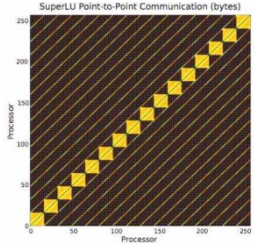
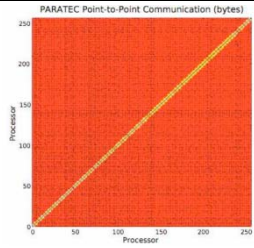
- 高性能计算的七个小矮人如何在更广泛的应用中捕获计算和通信模式？
- 除了高性能计算之外，还需要添加哪些小矮人来覆盖缺失的重要领域？

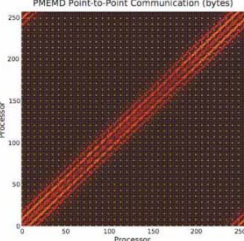
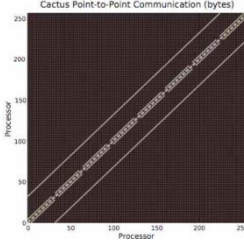
如果我们发现一组扩展的小矮人集是广泛适用的，我们就可以用它们来指导创新和评估的新模型。只要最终列表中不超过两打或三打小矮人，架构师和编程模型设计师就可以使用它们来衡量成功与否。相比之下，SPEC2006 有 29 个基准，EEMBC 有 41 个。理想情况下，我们希望整个小矮人集都有良好的表现，以表明新的多核架构和编程模型在未来的应用中表现良好。

小矮人是在高度抽象的层次上指定的，可以将相关但完全不同的计算方法分组。随着时间的推移，单个小矮人可以扩展为覆盖各种各样的方法，此时它应该被视为多个不同的小矮人。只要我们不以太多的小矮人结束，那么在囊括新的小矮人过程中犯错似乎更明智。例如，非结构化网格可以被解释为一个稀疏矩阵问题，但是这会将问题限制在一个单一的间接层面上，且会忽略该问题的很多附加信息。[【这一点确实如此。一般结构化或者非结构化网格本质上是求解稀疏线性](#)



代数系统，即可以形式化成求解  $Ax=b$ 。但是在很情况下，应用专业并不这样做。  
所以在小矮人把这两个小矮人与稀疏线性代数分开是合适的。】

<i>Dwarf</i> 小矮人	描述	通信模式(图的轴显示了1-256个处理器, 黑色表示无通信)	NAS 基准测试程序/ HW 例子
1. 稠密线性代数 (例如, BLAS [Blackford et al 2002], ScaLAPACK [Blackford et al 1996], or MATLAB [MathWorks 2006])	数据是稠密矩阵或者稠密向量。(BLAS Level 1 = vector - vector; Level 2 = matrix - vector; Level 3 = matrix - matrix) 这样的应用程序一般采用单位跨度存储器访问的方式从行中读取数据, 采取跨越访问的方式从列中读取数据。	 <p>MadBench 的通信模式是大量使用 ScaLAPACK 来实现并行稠密线性代数的典型代表, 它是更为广泛的数值算法。</p>	三对角矩阵, LU-SGS / 向量机, 阵列机
2. 稀疏线性代数 (例如, SpMV, OSKI [OSKI 2006], or SuperLU [Demmel et al 1999])	数据集包含许多零值。数据通常存储在压缩矩阵中, 以减少访问所有非零值的存储和带宽要求。一个例子是块压缩稀疏行 (BCSR)。由于压缩格式, 数据通常通过索引加载和存储进行访问。【CSR, 有非常多的存储格式】	 <p>SuperLU (上图中的通信模式) 使用 BCSR 方法来实现稀疏 LU 分解。</p>	共轭梯度带有收集分散的矢量计算机度 / 带有收集分散的矢量计算机【SpMV 是核心】
3. 谱方法 (例如, FFT [Cooley and Tukey 1965])	数据处于频域, 而不是时间或空间域。通常, 光谱方法使用多个蝶形阶段, 其将乘加运算和特定数据排列模式组合, 其中一些阶段为全部通信, 而另一些阶段严格为局部。	 <p>PARATEC: 3D FFT 需要一个全面的通信来实现 3D 传输, 这需要每个链路之间的通信。对角线条纹描述了正交化所需的 BLAS-3 主导线性代数步骤。</p>	傅里叶变换 / DSPs, Zalink PDSP [Zarlink 2006] 【FFTW】

4. N-Body 方法（例如，Barnes-Hut [Barnes and Hut 1986], Fast Multipole Method [Greengard and Rokhlin 1987]）	取决于许多离散点之间的相互作用。变化包括粒子-粒子方法，其中每个点都依赖于所有其他点，导致 $O(N^2)$ 的计算复杂性，还有分层粒子方法，这些方法结合了来自多个点的力或势能，以将计算复杂度降低到 $O(N \log N)$ 或 $O(N)$ 。	 <p>PMEMD 的通信模式是粒子网格 Ewald 计算。</p>	（没有基准测试程序） / GRAPE[Tokyo 2006], MD-GRAPE [IBM 2006]
5. 结构化网格（例如，Cactus [Goodale et al 2003] or LatticeBoltzmann Magnetohydrodynamics [LBMHD 2005]）	通过常规的网格表示；根据原理网格上的点一起更新。它具有很高的空间局部性。可能是合适的更新，也可能在两个不同版本的网格之间更新。网格可能再次分为更细的网格（“自适应网格细化”）；并且粒度之间的转换可能会动态发生。	 <p>Cactus 的通信模式，这是在三维块结构网格上使用 7 点模板的 PDE 解算器。</p>	多重网格，标量五边形 / QCDOC[Edinburgh 2006], BlueGeneL <b>【CFL3D】</b>
6. 非结构化网格（例如，ABAQUS [ABAQUS 2006] or FIDAP [FLUENT 2006]）	选择数据所在的不规则网格，通常由应用程序的基础特性决定。数据点位置和相邻点的连通性必须明确。原则上，网格上的点一同更新。更新通常涉及多个级别的内存引用间接寻址，因为任何点的更新都需要先确定相邻点的列表，然后从这些相邻点加载值。		非结构化自适应 / 带有收集分散的矢量计算机，Tera 多线程体系结构[Berry et al 2006] <b>【SU2，有限元等】</b>
7. 蒙特卡罗（例如，Quantum Monte Carlo [Aspuru-Guzik et al 2005]）	计算取决于重复随机试验的统计结果。考虑到了易并行。	在蒙特卡罗方法中，通信通常不占优势。	易并行 / NSF 万亿次网格 <b>【MCNP】</b>

表格 3. 7 个小矮人及其描述，相应的 NAS 基准和示例计算机

为了研究 7 个小矮人的一般适用性，我们将这个列表与其他的基准集合进行了比较：EEMBC（来自嵌入式计算），SPEC2006（来自桌面和服务器的计算）。这些集合独立于我们的研究，所以它们作为验证我们的小型计算内核是否是未来应用程序的好目标。我们将在第 3.5 节中详细描述最后的列表，但是我们从我们对 41

个 EEMBC 内核和 26 个 SPEC2006 程序的检查中，我们发现 4 个小矮人加入到这个列表中：

- 组合逻辑通常涉及对大量数据执行简单的操作，这些数据常常利用位级并行。例如，计算循环冗余码(CRC)对于确保数据安全的完整性和 RSA 加密至关重要。【很多纯计算机类操作在实际中发挥重大作用，如图像处理、深度学习和区块链等。】
- 图形遍历应用程序必须遍历许多对象，并检查这些对象的特性，例如将用于搜索。它通常包括间接表查找和小计算。【Graph500。】
- 图形化模型应用程序包括将随机变量表示为节点和条件依赖性的图。例如，包括贝氏网络和隐马尔可夫模型。【这个不太懂。】
- 有限状态机表示一组相互连接的状态，例如用于解析的状态。一些状态机可以分解成多个同时活动的状态机，它们可以并行运行。【这个也不太懂。】

为了超越 EEMBC 和 SPEC，我们研究了三个日益重要的应用领域，看看是否应该增加小的数量：机器学习、数据库软件、计算机图形和游戏。【确实是重要的应用领域。】

### 3.2.1 机器学习

使用统计机器学习来从大量数据中获得意义是未来计算领域最有希望的领域之一，这些数据来自于快速计算机、更大的磁盘，以及使用 Internet 将它们连接在一起。

迈克尔·乔丹(Michael Jordan)和丹·克莱因(Dan Klein)是我们在机器学习方面的专家，他们发现两个小矮人应该加入到支持机器学习的行列中：

- 动态规划是一种算法技术，可以计算解决简单重叠子问题的方法。它特别适用于优化问题，因为问题的最优结果是由子问题的最优结果建立起来的。【Dynamic programming 的翻译是动态规划，不是动态编程。】
- Backtrack 和 Branch-and-Bound：这些都涉及到解决各种各样的搜索和全局优化问题。排除搜索空间中没有有趣解决方案的区域，需要一些隐含

方法。分支限界算法的工作原理是分而治之：搜索空间被细分为更小的子区域(“分支”)，并在考虑到的每个子区域的所有解决方案中找到边界。

同样，还有许多其他著名的机器学习算法适用于现有的小矮人：

- 支持向量机[Cristianini and Shawe-Taylor 2000]：稠密线性代数。
- 主成分分析[Duda and Hart 1973]：稠密或稀疏线性代数，具体取决于实施的细节。
- 决策树[Poole et al 1998]：图遍历。
- 哈希：组合逻辑。【除了加解密外，在区块链和比特币中得到应用。不需要通信。】

### 3.2.2 数据集软件

微软研究院的 Jim Gray 认为排序是现代数据库的核心。他发起了一个竞赛，看谁能提出最快的分类器。在一分钟内尽可能整理最多的数据，组织成 100 字节的记录。2006 年的优选者在 32 路共享内存多处理器上，使用 1.6 GHz Itanium 2 处理器，128gb 的主存和 128 个磁盘，整理了 4 亿条记录(40 GB)。使用一个名为 Nsort 的商业排序包，它可以直接对记录进行排序，也可以对记录进行分类。[Nyberg et al 2004]排序算法是样本排序。虽然在 I/O 和主内存之间有有效的接口来快速排序大型文件，但排序并不会增加我们的小矮人列表。【这就是在学算法时，为什么要学排序算法，冒泡排序，快速排序等。】

现代数据库的另一个重要功能是哈希。与典型的散列不同，数据库散列将计算大量数据。同样，这些计算和通信模式不会扩展小矮人。【应该提下比特币，数字货币什么的。】

我们的数据库专家 Joe Hellerstein 认为，在互联网上对大数据进行收集是数据库的发展趋势。探索此类集合的关键是由谷歌开发并广泛应用的编程模型 MapReduce。[Dean and Ghemawat 2004]第一阶段将用户提供的函数映射到成千上万的计算机上，处理键/值对来生成一组中间键/值对，第二阶段减少将在成千上万的实例中，通过合并所有的中间键/值对生成的单一的结果。注意，这两个阶段是高度并行的，但理解起来却很简单。在 Lisp 中借用类似函数的名称，他们称之为最初的“MapReduce”。

MapReduce 是 Monte Carlo 的一个更普遍的版本：本质是一个单独的函数，它在独立的数据集上并行执行，输出最终组合成一个或多个结果。为了反应更大的作用域，我们将小矮人称为 MapReduce。

数据库未来的第二个推动力是遗传学。[Altschul et al 1990]BLAST 是一种广泛流行的启发式代码（基本的本地对齐搜索工具），用于寻找与数据库相似的 DNA/蛋白质序列的区域。有三个主要步骤：

1. 从序列中编译一个高分单词列表。
2. 扫描数据库以查看此列表中的匹配项。
3. 扩大命中以优化匹配

很明显，BLAST 并没有扩大我们的小矮人列表。

### 3.2.3 计算机绘图和游戏

人们竞相研究增强现实技术，推动了 GPU 性能的提升，现在已经到了 Teraflops 级别，但在屏幕上绘制多边形和纹理时，图形真实感并不理想。相反，管理这些图形对象行为的物理过程建模需要许多相同的用于大规模科学模拟的计算模型。计算机视觉和媒体处理中的许多任务也是如此，这些任务构成了硬件驱动供应商技术路线图的“未来应用”的核心。

对于所有实际目的而言，通过 GPU 采用片上并行机制来加速计算机图形处理是一个已经被解决的问题。此时主处理器的主要负担集中在对构成游戏或用户界面的图形元素的物理属性进行建模。真实的物理需要对物理过程进行计算建模，这与科学计算应用程序所需的一样重要。采用的计算方法非常类似那些激发七个原始小矮人的方法。

例如，电影中通过特效产生的液体和液体行为的建模通常使用像平滑粒子流体动力学（SPH）[Monaghan 1982]这样的粒子方法完成。物理模型的渲染仍然在使用 GPU 或软件渲染器在 OpenGL 中完成，但液体流动形状的基础模型需要基于粒子的流体模型。还有其他几个例子，在游戏和图形中将物理属性建模成其他的小矮人：

- 反向运动学需要稀疏矩阵计算和图遍历方法的组合。

- 弹簧模型用于模拟响应压力或冲击而变形的任何刚性物体，例如弹跳球或 Jell-0，使用稀疏矩阵或有限元模型。【Spring models 可以用于 CFD 等网格变形所用。】
- 碰撞检测是一种图遍历操作，就像用于深度分类和隐藏表面去除的 Octrees 和 Kdtrees 一样。
- 对碰撞的响应通常以有限状态机的形式实现。【没有搞过，不理解。】

因此，令人惊讶的结论是，游戏和绘图并没有超出上述列出的 13 种 drawfs 的限制。

从 GPU 和绘图软件得出的一个令人鼓舞的经验是，API 不直接暴露程序员的并发性。例如，OpenGL 允许程序员用 Cg（用于描述此类操作的专用语言）描述一组应用于场景中每个多边形的“顶点着色器”操作，而不必考虑有多少硬件碎片处理器或顶点处理器在 GPU 的硬件实现中可用。【Cg 是 GPGPU 最先开始的开发语言之一。】

### 3.2.4 接下来的六个小矮人的总结

图 4 显示了上一节中由于研究而增加的六个小矮人。请注意，我们认为算法独立于数据大小和类型（请参阅第 5.3 节）。

Dwarf	Description
8 组合逻辑（例如，加密）	用逻辑功能和存储状态实现的功能。【组合逻辑问题很常见，如 TSP、VRP 问题，规模大了后存在组合爆炸的问题。】
9.图遍历（例如，快速排序）	通过沿着连续的边来访问图中的许多节点。这些应用程序通常涉及很多级别的间接寻址和相对较少的计算量。 【甘老师最爱的 Graph500，在国防科大 E 级验证系统上得到结果应该可以排进 2017 年榜单的前 10 名。】
10.动态规划	通过解决更简单的重叠子问题来计算解决方案。特别适用于有大量可行解决方案的优化问题。
11. 回溯和 Branch and	通过递归将可行区域划分为子域，然后精简次优子问题



Bound (分支限界)	来找到最优解。
12.构建图形模型	构造将随机变量表示为节点并将条件依赖性表示为边的图。例子包括贝叶斯网络和隐马尔可夫模型。
13. 有限状态机	一种行为由状态定义的系统，由输入和当前状态定义的转换以及与转换或状态相关的事件。 <a href="#">【游戏？不太了解。】</a>

图 4. 原 7 个小矮人的扩展

尽管 13 个小矮人中有 12 个具有某种形式的并行性，但有限状态机（FSM）看起来是一个挑战，这就是为什么我们让它成为最后一个小矮人的原因。就像 MapReduce 是高度并行一样，FSM 可能会被证明是“embarrassingly sequential”。如果它仍然很重要，并且不能并行，那将是令人失望的，但也许正确的永久解决方案是改变算法。在多核和众核时代，顺序计算时代的流行算法可能会逐渐衰退。例如，如果霍夫曼解码被证明是“embarrassingly sequential”，那么可能我们应该使用一种不同可以并行化的压缩算法。

无论如何，13 个小矮人的重点不在于找出高度并行这种可轻松实现的目标。关键是要确定未来十年内与并行性无关的重要应用程序的核心计算和通信内核。为了开发能够尽可能有效运行未来应用程序的编程系统和架构，我们必须了解这些限制和机遇。然而，我们注意到，易并行代码的低效率可能是未来架构失败的原因。更多的小矮人可能需要添加到列表中。然而，我们感到惊讶的是，我们只需要增加六个小矮人来覆盖如此广泛的重要应用。

### 3.3 小矮人的组成部分

任何重要的应用程序（如 MPEG4 解码器或 IP 转发器）都将包含多个小矮人，每个小矮人都占用应用程序计算量的相当大的百分比。因此，大型应用程序的性能不仅取决于每小矮人的表现，还取决于小矮人如何在平台上组合在一起。

包含应用程序的小矮人的集合可以通过两种不同的方式分布在多处理器平台上：

1. 在一组通用处理器上进行时间分布或时间共享，或者
2. 空间上的分布或共享空间，每个小矮人单独占用一个或多个处理器。

时间或空间分布的选择部分取决于小矮人之间的通信结构。例如，有些应用程序的结构是一系列的串行阶段，其中每个阶段都是一个小矮人，必须在我们开始下一个阶段之前完成。在这种情况下，使用时间复用将整个处理器分配到每个阶段是很自然的。其他的应用程序可以被构造成一个同时运行的通信小矮人网络，在这种情况下，在可用处理器上空间分布小矮人是很自然的。

以上两种分布形式可以分层应用。例如，一个小矮人可以被实现为一个 **pipeline**，在这个 **pipeline** 中，输入被划分为多个阶段，每个阶段都运行在自己的处理器的空间上。每个阶段都是跨连续输入的时间多路复用，但是通过 **pipeline** 阶段的空间分布来处理单个输入流。

考虑小矮人的组成时出现两个软件问题：

1. 组合模型的选择——如何将小矮人组合在一起形成一个完整的应用程序。科学软件界最近开始转向组件模型[Bernholdt et al 2002]。然而，在这些模型中，单个模块的耦合并不紧密，这可能会影响最终应用程序的效率。【组件模型是啥？不了解。】

2. 数据结构转换。各种算法可能有自己的首选数据结构，如稠密矩阵的递归数据布局。这可能与组合的效率不一致，因为工作集必须在其他小矮人使用之前进行转换。

这些问题是一个更大的难题。描述可组合并行代码库的有效方法是什么？我们是否可以编写一个库，以便在完全并行的应用程序中与其他库组合时，对其理想映射的知识进行编码？如果理想映射严重依赖于编译时未知的输入数据，该怎么办？

### 3.4 英特尔的研究

英特尔认为，对计算需求的增加将来自于处理“Era of Tera”所提供的海量信息。[Dubey 2005]英特尔将计算分为三类：识别、挖掘和合成，缩写为 **RMS**（**Recognition, Mining, and Synthesis**）。识别是机器学习的一种形式，其中计算机检查数据并构造该数据的数学模型。计算机构造模型后，挖掘将搜索网络查找该模型的实例。合成是指创建新模型（如在图形中）。因此，**RMS** 与第 3.2.1 至 3.3.3 节中我们对机器学习，数据库和图形的检查相关。



RMS 的通用计算主题是 "大型复杂数据集的多模式识别和合成" [Dubey 2005]。英特尔相信 RMS 将在医药、投资、商业、游戏和家庭中找到重要的应用。英特尔在图 5 中所做的努力表明，Berkeley 并不是唯一一个试图将计算新领域组织到底层计算内核以指导架构研究的人。

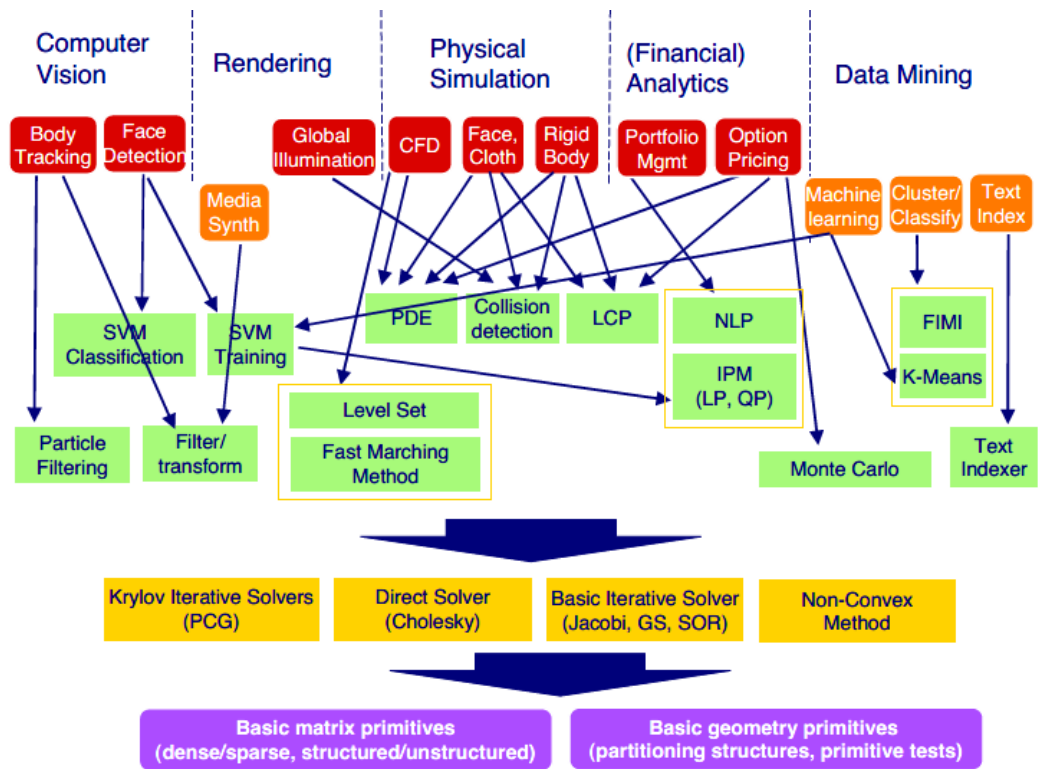


图 5. 英特尔的 RMS 以及它如何映射到更原始的函数。在这个上图顶部的五个类别中，计算机视觉被分类为识别，挖掘（数据挖掘），以及合成（渲染、物理模拟、财务分析）。[Chen 2006]。

### 3.5 13 个小矮人

图 6 总结了我们的调查，并展示了 13 个小矮人的存在，包括 EEMBC、SPEC2006、机器学习、图形/游戏、数据库软件和英特尔的 RMS。如上所述，一些程序使用多个小矮人，因此它们被列在多个类别中。我们相信我们的小矮人名单尚未完成，我们预计未来还会增加更多的小矮人。与此同时，我们感到惊讶的是，一组不同的重要应用得到了小部分小矮人的支持。

小矮人	嵌入式计算	通用计算	机器学习	图形/游戏	数据库	英特尔 RMS
1. 稠密线性代数(例如: BLAS 或 MATLAB)	EEMBC Automotive: iDCT, FIR, IIR, Matrix Arith; EEMBC 使用者: JPEG, RGB to CMYK, RGB to YIQ; EEMBC 数字娱乐: RSA MP3 解码, MPEG-2 解码, MPEG-2 编码, MPEG-4 解码; MPEG-4 编码; EEMBC 网络: IP 数据包; EEMBC 办公自动化: 图像旋转; EEMBC 电信: 卷积编码; EEMBC Java: PNG	SPEC 整数: 量子计算机模拟 (libquantum), 视频压缩 (h264avc) F1 范式: 隐藏马尔科夫模型 (sphinx3) <a href="#">【电磁学, 边界元】</a>	支持向量机, 主成分分析, 独立分量分析		数据库 哈希访问大量的连续内存段	人体跟踪, 媒体综合 线性规划, Kmeans, 支持向量机, 二次规划, PDE: Face, PDE: Cloth*

2. 稀疏线性代数(例如: SpMV, OSKI, 或 SuperLU)	EEMBC Automotive: Basic Int + FP, Bit Manip, 可远程数据, 表查找, Tooth to Spark; EEMBC 电信: 位分配; EEMBC Java: PNG	F1 范式: 流体力学 (bwaves), 量子化学 (gameess; tonto), 线性规划求解 (soplex)	支持向量机, 主成分分析, 独立分量分析	反向运动学; 弹簧模型		支持向量机, 二次规划, PDE: Face, PDE: Cloth* PDE: 计算流体力学
3. 谱方法 (例如, FFT)	EEMBC Automotive: FFT, iFFT, iDCT; EEMBC 使用者: JPEG; EEMBC 数字娱乐: MP3 解码		谱聚类	纹理映射		PDE: 计算流体力学 PDE: Cloth
4. N-Body 方法 (例如, Barnes-Hut, 快速多极算法)		F1 范式: 分子动力学 (gromacs, 32-bit; namd, 64-bit)				

5. 结构化网格 (例如, Cactus 或 格子玻尔兹曼流体力学)	EEMBC Automotive : FIR, IIR; EEMBC 使用者 : HP GrayScale; EEMBC 使用者: JPEG; EEMBC 数字娱乐: MP3 解码, MPEG-2 解码, MPEG-2 编码, MPEG-4 解码; MPEG-4 编码; EEMBC 办公自动化: 抖动; EEMBC Telecom: 自相关	F1 范式: 量子动力学 (推理), 磁流体力学 (zeusmp), 广义相对论 (cactusADM), 流体动力学 (leslie3d-AMR ; lbm), 有限元方法 (dealIII-AMR; calculix), 麦克斯韦方程组求解 (GemsFDTD), 量子晶体学 (tonto), 天气建模 (wrf2-AMR) <b>【流体力学】</b>		平滑 ; 插值		
6. 非结构化网格 (例如, ABAQUS 或 FIDAP)		<b>【流体力学, 结构力学】</b>	信仰传播			全局照明
7. MapReduce		F1 范式.: 射线追踪器 (povray)	期望最大化		MapReduce	

( 例 如 , 蒙 特 卡 洛)		【MCNP】				
8. 组 合 逻辑	EEMBC 数字娱乐: AES, DES EEMBC 网络: IP 数据包, IP NAT, 路由查 找; EEMBC 办公自 动化: 图像旋 转 EEMBC Telecom: 卷 积编码	【TSP、VRP】	哈希		哈希	
9. 图 遍 历	EEMBC Automotive : 指 针 雕 镂 , Tooth to Spark; EEMBC 网络: IP NAT, OSPF, 路由查 找 ;EEMBC 办 公自动化: 文 本处理; EEMBC Java: 国际象 棋, XML 处理		贝叶斯 网 络 , 决策树	反向运 动 学 , 碰撞检 测, 深 度 排 序, 隐 藏表面 去除	传递闭 包	自 然 语 言处理
10. 动 态 规划	EEMBC Telecom: 维	SPEC 整数: Go (gobmk)	前 向 向 后、内		查询优 化	

	特比解码		外部、 可变消 除、值 迭代			
11. 后轨 和分支限 界		SPEC 整数： 国 际 象 棋 (sjeng), 单 纯 形 网 络 算 法 (mcf), 2D 路径 查找库 (astar)	核 回 归, 约 束满足			
12. 图形 化模型	EEMBC  Telecom: 维 特比解码	SPEC 整数: 隐 马尔可夫模型 (hmmr)	隐马尔 可夫模 型			
13. 有限 状态机	EEMBC  Automotive : 角度到时间, 缓存 “克星”, 可以远程数 据, PWM, 道路 速度, Tooth to Spark; EEMBC Consumer : JPEG; EEMBC 数字娱 乐: 哈夫曼编 码, MP3 解码, MPEG-2 解码, MPEG-2 编码,	SPEC 整数: 文 本 处 理 (perlbench), 压 缩 (bzip2), 编译器 (gcc), 视 频 压 缩 (h264avc), 网 络离散事件模拟 (omnetpp), XML 转 换 (xalancbmk)  【看起来很重 要, 只是没有搞 过。】		对碰撞 的响应		

	MPEG-4 解码; MPEG-4 编码; EEMBC 网络; QoS, TCP; EEMBC 办公自 动化: 文本处 理; EEMBC Telecom: 位 分配; EEMBC Java: PNG					
--	---	--	--	--	--	--

图 6. EEMBC、SPEC2006、机器学习、图形/游戏、数据库和英特尔 RMS 映射到 13 个小矮人。请注意，SVM，QP，PDE：Face 和 PDE：Cloth 可以使用稠密矩阵或稀疏矩阵，具体取决于应用。

## 4.0 硬件

现在，我们已经给出了对于图 1 左侧方框中关于并行计算的应用程序和小矮人的观点，接下来准备探讨右边方框的部分：硬件。上面的第 2 章描述了关于半导体工艺当前以及未来的制约因素，但同时也提供了许多机会。

我们围绕 30 多年前首次用于描述计算机的三个组件的硬件部分进行了研究，它们分别是：处理器、内存和交换[Bell and Newell 1970]。

### 4.1 处理器：小而美

在诸如钢铁制造业等许多现代技术的发展中，我们可以观察到在很长的一段时期里，更大的等同于更好的。要辨认这些发展时期很简单：一个顶尖的工程示例只是被一个更好的所取代。由于规模经济或其他经济因素的减少，这些技术的发展不可避免地发生了转折，继而永远地改变了其发展的道路。我们认为通用微处理器的发展正在触及这样一个转折点。

第 2 部分中的第 4 条新 CW（传统智慧）指出，我们可以成功设计和制造的尺寸规模正在不断缩小。而新 CW 的第 2 部分的第 1 条和第 2 条指出，功率是当前和未来几代处理单元的主要约束。**【功耗确实是个更大的问题，高效能计算应该是未来后 E 级计算的发展趋势。】**为了支持这些说法，我们注意到，多个下一代处理器（例如 Intel 的 Tejas Pentium 4 处理器）都因为功耗问题被取消或重新定义了[Wolfe 2004]。即便是定位于“时钟周期越短越好”的 Intel 公司也宣称，通过最大化时钟速度来实现性能最大化的传统方法已经达到了极限[Borkar 1999][Gelsinger 2001]。在本节中，我们将抛开转折点来看一个问题：什么处理器是构建未来多处理器系统的最佳部分模块？

用更小的处理模块来构建未来的微处理器系统有许多优点：

- 并行性的实现是一种高效的节能途径[Chandrakasan et al 1992]。
- 多个小型内核为并行代码提供了较高的单位面积性能。
- 大量的小型处理部件能够更细粒度地进行电压的动态缩放和断电。
- 小型处理单元是一种经济的单元部件，在面临灾难性缺陷时易于关闭，并且在面对大参数变化时也更容易重新配置。Cisco Metro 芯片[Eatherton 2005]为每



个芯片增加了四个冗余处理器，而 Sun 公司以 8-处理器的设计为基础，销售 Niagara 版本的 4-处理器、6-处理器或 8-处理器。据报道，图形处理器也正以这种方式使用冗余处理器，在索尼的 Playstation 3 游戏机中，8 个协同处理器中只有 7 个使用了 IBM Cell 微处理器。

- 具有简单架构的小型处理元件更易于设计并且更易于功能验证。尤其是与乱序执行的复杂体系结构相比，它更适合于正式的验证技术。【确实如此，看看 GPU。】

- 单个较小的硬件模块更加节能，其性能和功耗特性在现有的电子自动化设计流程中也更易于评估检测 [Sylvester and Keutzer 1998][Sylvester and Keutzer 2001] [Sylvester et al 1999]。

虽然上述观点表明我们应该考虑采用更小的处理器架构来实现我们的基本模块，但它们并不能准确指出哪种电路规模或处理器架构能够为我们提供最好的服务。我们认为，我们必须摆脱简单的“更大即更好”的方法。但是，这也并不意味着“最小的就是最好的”。

#### 4.1.1 什么处理单元最合适？

确定最佳的处理单元模块需要依赖于应用程序，部署环境，工作负荷，目标市场约束以及制造技术等多元优化问题的解决方案或初步的解决方案。然而，显而易见的是，性能和功耗之间的权衡对于当前和未来的多处理器系统的整个系统应用来说都是至关重要的。【这一段话没有毛病。】

区分能量（焦耳）和功率（焦耳/秒或瓦特）是很重要的，功率是指消耗能量的速度。在设计当中，每个任务消耗的能量通常需要尽量最小化，而峰值功耗则通常被视为设计约束。处理器计算过程中消耗的能量会影响移动设备的电池寿命以及为服务器群供电的成本。峰值功率决定了封装和冷却处理器的成本，并且这些成本将随着急剧上升的功率而迅速增长。必须限制芯片温度以避免过多的功率泄漏。由于电迁移和其他高温可靠性问题，芯片温度过高还可能导致寿命缩短。对于风冷式服务器和台式机芯片来说，其峰值功耗的合理上限是 150W，而对于笔记本电脑而言是 40W，低成本/低功耗的嵌入式应用芯片是 2W。

不同的应用程序在性能和能耗之间在不断的进行权衡。例如，许多实时任务

（例如，在笔记本电脑上观看 DVD 电影）具有固定性能需求——寻求最低的能量实现。桌面处理器通常在最大的功率限制下寻求最高性能。请注意，如果设计的处理单元不能在耗尽预定可用的电力前，足够快地完成任务，那么在功率限制下，即使设计中的每个操作都消耗最少的能量，这个设计也可能无法实现最高的性能。

如果所有任务都是高度可并行化的并且硅片面积是冗余的，那么我们会希望每条指令在内核中消耗的能量最低（SPEC /Watt）。然而，我们也需要并行代码较少并且芯片单位面积的高吞吐量来降低成本。我们面临的挑战是在提高性能的同时，不会增加每次操作的能耗或硅片面积。

[Gonzalez and Horowitz 1996]研究了微体系结构对能耗和延迟的影响。使用能耗延迟积（ $\text{SPEC}^2/\text{W}$ ）作为度量标准，作者确定，简单的流水线技术对于解决延迟有显著效果，但同时也会在一定程度上增加能耗。相反，超标量体系结构会对能耗延迟积产生不利影响。硬件所需的额外功率开销并未超过其性能优势。指令级的并行性是有限的，所以通过广泛性和预测执行等技术来设计的微架构适度的提升了运算性能，但却牺牲了大量的硅片面积和能耗开销。【悖论是处处存在的，受限制的优化是面临的普通问题。】

许多研究人员[Hrishikesh 等人 2002] [Srinivasan 等人 2002] [Harstein 和 Puzak 2003] [Heo 和 Asanovic 2004]已经研究了微体系结构中流水线级数的最佳数量。这些结果在[Chinnery 2006]中进行了总结和审查。请注意，迄今为止，单处理器基准（如 SPEC）已成为测量计算和能效的最常用基准。我们相信未来的基准测试集必然会发展，并且会出现更具代表性的混合应用，包括基于 13 个小矮人的并行代码，避免单线程性能的过度优化。由于上述结果依赖于制作工艺技术，逻辑族，基准集以及工作负载，因此很难将结果推广到我们的目的域。然而，Horowitz [Horowitz 2006]，Paulin [Paulin 2006]等文献综述以及我们自己的调查 [Chong 和 Catanzaro 2006]显示，对现有体系结构的经验数据进行分析后表明，有序执行的浅层流水线已被证明具有最大面积利用率和能效。【确实如此。】鉴于对这些物理和微观架构的考虑，我们相信未来高效体系结构的构建模块可能是简单的，包括适度流水线（5-9 级）处理器，浮点单元，矢量和 SIMD 处理单元等。请注意，这些限制违背了使用最大可用处理器来简化并行编程的传统观点。

### 4.1.2 我们真的能在一块经济的芯片上安装 1000 个内核吗？

未来基本处理器构建模块的大小和复杂性的显著降低意味着可以在单个裸片上经济地实现更多的内核；此外，这个数字可以随着硅片的换代而成倍增加。例如，“manycore”级数很可能是 128、256、512 .....内核，而不是在相同的半导体工艺时代下，当前的“multicore”计划中的 2、4、8 .....内核。

有强有力的经验证据表明，当 30nm 级的技术投入应用时，我们可以在芯片上实现 1000 个内核。（由于英特尔已经推出了 45nm 技术芯片，未来 30nm 并不遥远。）思科在其路由器中嵌入了一个网络处理器，其中包含以 130nm 级技术实现的 188 个内核。[Eatherton 2005]该芯片的尺寸为  $18\text{mm} \times 18\text{mm}$ ，在 250MHz 时钟频率下功耗为 35W，每秒产生 500 亿条指令。单个处理器内核是具有非常小的 5 级缓存 Tensilica 处理器，每个内核的尺寸为  $0.5\text{mm}^2$ 。大约三分之一的芯片是 DRAM 和特定的功能单元。简单地按照摩尔定律进行扩展，就可以得到 45nm 技术下的 752 个处理器和 30nm 技术下的 1504 个处理器。不幸的是，功耗可能不会随着尺寸的减小而减少，但是在突破桌面或服务器应用的 150W 限制之前，我们有足够的空间。

### 4.1.3 是否有一种处理器适用于任何场合？

我们简要地考虑一下这个问题，即将来的多处理器是作为相同处理器的集合来构建，还是由不同的异构处理单元组装而成。现有的嵌入式多处理器（例如 Intel IXP 处理器系列），在芯片上保留了至少一个通用处理器，用来支持各种内部的管理功能，并为一般的（例如 Linux）操作系统提供基础硬件支持的功能。同样，IBM Cell 也有一个通用处理器和八个定制处理单元。在芯片上保留更大的处理器有可能利于加快“固有顺序”代码段的运行速度或者减少工作负载的线程数量 [Kumar et al 2003]。

正如 Amdahl 在 40 年前观察到的那样，一个程序中的并行部分可能会限制并行计算机的性能 [Amdahl 1967]。因此，在多核体系结构中拥有不同“大小”的处理器的一个原因是通过减少运行并行代码的时间来提高并行速度。例如，假设在 100 个处理器计算机上，一个程序有 10% 的时间没有加速。假定以两倍的速度

运行顺序代码，由于更大的功耗预算，更大的缓存，更大的乘法器等等，单个处理器需要的资源是单核运行时所需资源的 10 倍，它值得吗？使用 Amdahl 定律 [Hennessy 和 Patterson 2007]，与单个简单处理器相比，同构 100 个简单处理器设计和异构 91 处理器设计的相对加速比为：

$$\text{SpeedupHomogeneous} = 1 / (0.1 - 0.9 / 100) = 9.2 \text{ 倍}$$

$$\text{SpeedupHeterogeneous} = 1 / (0.1 / 2 - 0.9 / 90) = 16.7 \text{ 倍}$$

在这个例子中，一个较大的单处理器需要 10 倍的资源来获得两倍的运行速度，因此把它替换为 10 个更小的处理器更有价值。

异构处理器解决方案不仅可以帮助运用 Amdahl 定律，还可以在功耗、延迟和面积方面表现出显著优势。异构处理器实现的好处是能够最小化软件开发的成本同时也使得处理器指令集具有可配置性 [Killian 等人, 2001]，但这需要对每种新设计实行定制以实现性能优势，而这仅仅对于大型市场来说在经济上是合理的。

#### 【异构处理器编程太麻烦了。】

另一种在同构结构中实现异构的方法是在预定义的可重新配置的逻辑结构中实现定制的软处理器。然而，当前区域（40 倍），功耗（10 倍）和延迟（3 倍）开销 [Kuon 和 Rose, 2006] 似乎使这种方法对于通用处理而言过于昂贵。一种有支持处理器异构的方法是添加一个可重新配置的协处理器作为一个单独的芯片 [Hauser 和 Wawrzynek, 1997] [Arnold, 2005]。这消除了对新定制芯片的需求。目前的数据还不足以确定这种方法作为节能的解决方案。

另一方面，单个复制处理单元具有许多优点，特别是它提供了易于实现的芯片和常规软件的环境。在具有数千个线程的环境中加入异构性可能会引起难以解决的问题。

与同构多核的灵活性和软件优势相比，异构多核可能拥有的功耗和面积优势会胜出吗？或者说，未来的处理器能否像晶体管一样，可以编织成任意复杂电路的单个构建模块？又或者说，处理器能否变得更像是一个标准单元库中的 NAND 门，成为数百个密切相关但独特的设备系列中的一个实例？在本节中，我们无法声称已经解决了这些问题。相反，解决这些问题肯定需要大量的研究和实验，而且这项研究的需求比业界多核多处理器路线图的研究更迫切。

## 4.2 解除访存限制

在过去的几十年里，DRAM 产业的发展大幅度降低了单位内存的价格，从 1980 年的每 GB 一千万美元[Hennessy 和 Patterson,2007]降至现在（2006 年底）的每 GB 一百美元。遗憾的是，正如第 2 部分的 CW #8 所指出的那样，处理器访存周期也从 1980 年的几个处理器周期大幅增至现在的几百个。此外，存储墙已成为获得良好性能的主要障碍（参见第 8 节图 9）。Thomas Sterling 在 SC06 会议上向专家小组成员提出这一针对性问题来表达这种担忧：“多核心最终是否会被存储墙扼杀？”[Sterling,2006]。【存储墙一个方面是众核对访存带宽的需求，另一方面是众核的访存冲突。第二个方面目前可能很少有人注意，以后也会是一个严重的问题。】

好消息是，如果我们看一下 DRAM 芯片的内部，我们会看到许多独立的大位宽存储块。[Patterson 等 1997]例如，一个 512Mbit DRAM 由数百个存储块组成，每个存储块的位宽达到几千位。显然，DRAM 芯片内有巨大的潜在带宽等待被开发利用，而且 DRAM 芯片内部的存储器延迟显然要优于分立的芯片互联时的延迟。

在为并行计算硬件创建新的硬件基础时，不能总假设主存是由分立 DRAM 芯片通过标准接口连接构成的，因为这会限制我们创新。新的封装技术，如 3D 堆叠技术，可能会大幅度提高访存带宽，减少访存延迟和功耗。尽管在有几千个处理器和数百个 DRAM 存储块的一般情况下我们无法避免全局通信，但某些重要的计算几乎完全是访问局部存储，因而可以从创新的存储器设计中受益。【面向应用的芯片设计？】

进行内存技术创新的另一个原因是越来越多的硬件成本正在从处理转向内存。旧的 Amdahl 定律指出，一个平衡的计算机系统的处理能力每提升 1 MIPS 就需要增加 1 MB 的主存容量[Hennessy 和 Patterson 2007]。

尽管 DRAM 容量与摩尔定律保持同步，在 1980 年至 1992 年期间每三年翻两番，但从 1996 年到 2002 年，速度降为每两年翻一倍。今天我们仍然使用 2002 年推出的 512Mbit DRAM。多核的设计将在单个芯片中释放更多的 MIPS。鉴于目前内存容量增长缓慢，这种 MIPS 爆发增长意味着将来存储面积占硅片总面积的比例会增大。

## 4.3 互联网络

在物理硬件互联层面，多核最初采用了内核和高速缓存间的总线或交叉开关，但这种解决方案不能扩展到数以千计的核上。我们需要片上拓扑结构与系统尺寸成线性关系，以防止互联过度复杂而占据多核系统更多的成本。可扩展的片上互联网络借鉴了大规模分组交换网络的想法[Dally and Towles 2001]。投片生产的芯片，例如 IBM Cell 采用多个环形网络来互联芯片上的九个处理器，并使用软件管理的内存在内核之间进行通信，而不是使用传统的缓存一致性协议。

尽管已有对统计流量模型的研究来帮助完善片上网络设计[Soteriou et al 2006]，但我们相信对 13 个小矮人的研究可以使我们洞察更广泛应用下的通信拓扑结构和资源需求。基于对覆盖全部小矮人的现有大量并行科学应用的通信需求的研究[Vetter and McCracken 2001] [Vetter and Yoo 2002] [Vetter and Mueller 2002] [Kamil et al 2005]，我们对以下片上通信需求进行探究，以开发更高效和按用户规格定制的解决方案：

- 聚合通信需求与点对点需求有很大区别。需要全局通信的聚合通信倾向于涉及非常小的主要是延迟限制的消息。随着核心数量的增加，这些细粒度，小于缓存行大小的聚合同步结构的重要性可能会增加。由于延迟提高可能比带宽缓慢得多（请参见第 2 节中的 CW #6），关注点分离意味着增加专用于集体的单独的延迟导向网络。他们已经出现在先前的 MPP 中。[Hillis and Tucker 1993] [Scott 1996]作为最近的一个大规模示例，IBM BlueGene/L 除了为点对点消息提供更高带宽的“Torus”互联网络外，还为集合操作提供了“树状”网络。这种方法对于采用 1000 以上的内核芯片互联实现可能是有益的。【异构网络？可能，但是会提高复杂性。】
- 大多数点对点消息的大小通常足够大，即使对于片上互联，它们仍然保持带宽限制。因此，每个点对点消息都会优先选择通过互联的专用点对点路径，以最大限度地减少网络结构内争用的可能性。因此，虽然通信拓扑不需要非阻塞交叉开关，但片上网络应具有较高的总带宽，并支持将消息流准确映射到片上互联拓扑。
- 研究表明，大多数点对点通信是稳定和稀疏的，并且主要受带宽的限制。除了 3D FFT（参见图 2）之外，点对点消息传递需求倾向于通过完全连



接的网络交换结构（例如交叉开关或胖树）仅使用一小部分可用通信路径。对于片上互联，考虑到资源需求规模为互联处理器内核数量的平方，对于大多数应用需求来说，非阻塞交叉开关可能是过度设计，否则将会浪费硅片面积。没有表现出“谱方法”小矮人的通信模式的应用【即类似 FFT】，用于片上互联的低互联拓扑可能会证明更多的空间和功率效率。

- 尽管通信模式被认为是稀疏的，但它们不一定等同于低度固定拓扑互联，如环、网格或超立方体。因此，为每个点对点消息传输分配专用路径并不能由任何固定级别的互联拓扑简单解决。为此，你可能需要仔细安排作业，以便它们匹配互联结构的静态拓扑，或是采用可重新配置以符合应用通信拓扑的互联结构。

目前观察到的通信模式与潜在的通信/计算模式密切相关。考虑到 13 个小矮人，互联可能需要针对相对有限的一组通信模式。并且编程模型可能会提供更高层次的抽象来描述这些模式。

对于带宽受限通信路径，我们希望采用一种方法来尽量减少交换机占用的芯片面积，同时符合应用通信拓扑的需求。优化互联拓扑以满足应用需求的直接方法是使用电路交换来增强分组交换，以重新配置交换之间的布线拓扑来满足应用通信需求，同时保持分组交换机提供的复用/解复用能力。该问题的逆向依赖于软件管理任务映射和任务迁移，以适应较低程度的静态互联拓扑。电路交换方法为重新配置互联拓扑提供了更快的方式，这对于具有快速变化/自适应通信需求的应用而言可能非常重要。在这两种情况下，运行时性能监测系统（参见 4.6 节），编译时间代码的仪器来推断通信拓扑需求或自调优（见第 6.1 节）都将在推断最佳互联拓扑和通信时间表方面发挥重要作用。

可以使用较简单的电路交换来提供专用线路，以使互联适应运行时应用的通信模式。该问题的一种可能解决方案是将分组交换与光电路交换组合进行混合设计。[Kamil 等人,2005] [Shalf 等人,2005]。然而，在微观尺度上，包含能适应通信拓扑的电路交换的混合交换设计可能能够满足所有未来的并行应用。混合电路交换方法可以通过定制运行时重新配置互联拓扑来消除未使用的电路路径和交换容量，从而为多核处理器带来更简单和占用面积更小的片上互联。

## 4.4 通信原语

最初的应用可能将多核和众核芯片简单地视为传统的对称多处理器(SMP)。但是, 芯片级多处理器(CMP)提供了与 SMP 完全不同的功能, 而且它提供了一些重大的新机遇:

- CMP 上的核心间带宽可能是 SMP 典型带宽的数倍, 由此可以终止性能瓶颈。
- 核心间延迟远低于 SMP 系统的典型值(至少一个数量级)。
- CMP 可以提供只在同一芯片上的核心之间进行操作的新的轻量级一致性和同步原语。这些栅栏的语义与我们在 SMP 上使用的语义完全不同, 并且将以低得多的延迟进行操作。

如果我们简单地将多核芯片视为传统的 SMP, 甚至是通过移植 MPI 应用程序(参见第 5 节中的图 7), 那么我们可能会错过一些非常有趣的, 可以利用这些新功能的新体系结构设计和算法设计的机会。【硬件级别的通信原语有些意思。】

### 4.4.1 一致性

常规 SMP 使用高速缓存一致性协议来提供内核之间的通信, 并且在一致性方案之上建立互斥锁以提供同步。众所周知, 标准一致性协议对于某些数据通信模式(例如生产者 - 消费者流量)是低效的, 但这样的低效会随着 CMP 核数的增加、潜在核心带宽的巨大提升以及延迟的降低而被放大。因此我们需要更灵活甚至可重新配置的数据一致性方案来利用提升的带宽和降低的延迟。例如大型的片上缓存, 它们可以灵活地适应私有配置或共享配置。此外, 由于实时嵌入式应用程序倾向于对存储器层次结构进行更直接的控制, 所以可以从被配置为软件管理便笺存储器的片上存储中受益。【可编程 Cache 是未来的发展方向, 类似 GPU 上的 Shared Memory。】

### 4.4.2 使用锁进行同步

处理器间的同步可能是在改进性能和可编程性方面最具潜力的领域。处理器



同步有两种类型：互斥和生产者-消费者。对于互斥而言，在众多竞争的并发活动中，一次仅允许一个活动更新一些共享的可变状态，但通常情况下，顺序并不重要。对于生产者-消费者同步而言，消费者必须一直等待，直到生产者生成了所需的值为止。常规系统通过锁来实现上述两种类型的同步。（传统 SMP 中也通常通过锁来处理多消费者和多生产者同步的障碍）。

众所周知，这些锁定方案很难编程，因为程序员必须记住将锁与每一个关键的数据结构关联起来，并且使用一个防死锁的锁定方案来访问这些锁。锁定方案本质上是不可兼容的，因此不能形成通用并行编程模型的基础。更糟的是，这些锁定方案是使用轮转等待来实现的，这将导致过度连贯的通信和处理器能力的浪费。尽管可以通过使用中断来避免轮转等待，但当前操作系统的硬件间中断和上下文切换使得这在大多数情况下是不切实际的。

#### 4.4.3 使用事务内存进行同步

互斥同步的一个可能的解决方案是使用事务存储器[Herlihy and Moss 1993]。多个处理器在事务内部推测更新共享内存，并且只会在事务成功完成而不与其他处理器冲突的情况下提交所有更新。否则，更新将被撤消并且回滚到事务的开始。事务模型实现了非阻塞互斥同步（在互斥锁或屏障上不存在阻塞）[Rajwar and Goodman 2002]。事务性内存简化了互斥，因为程序员不需要分配和使用显式的锁变量或担心死锁。【Transactional Memory 火了一段时间，现在搞的人少了。】

事务的连贯性和一致性（TCC，Transactional Coherence & Consistency）计划[Kozyrakis and Olukotun 2005]提议全球应用事务来取代传统的缓存一致性协议，并且当消费者到达生产者之前，通过推测回滚来支持生产者-消费者同步。

事务性内存是一个很有前途但仍很活跃的研究领域。目前的纯软件方案具有较高的执行时间开销，而纯硬件方案或缺乏通用语言支持所需的设施，或需要非常复杂的硬件。某些混合硬件软件方案很可能会出现，但在对这种方案的功能需求进行充分理解之前，还需要更多的使用事务性内存的实践经验。

#### 4.4.4 在内存中使用全空位（Full-Empty Bits）进行同步

减少生产者-消费者同步的开销将要允许更细粒度的并行化，从而在应用程序中提高可利用的并行性。早期的提案包括内存字节中的全空位，这些技术可能得在多核时代重新被审视[Alverson et al 1990] [Alverson et al 1999]。全空位已被证明有助于实现高效的大规模并行图算法（对应于下面的“dwarf”图），这对新兴的生物信息学，数据库和信息处理应用至关重要[Bader 和 Madduri2006]。值得注意的是，Jon Berry 等人(Berry et al. 2006)最近的研究表明，在一个简单的 4 核处理器 MTA 上运行的，为全空比特提供硬件支持的图形处理算法，它的性能优于 2006 年的 Top500 列表中最快的系统——64k 处理器的 BG/L 系统。【不懂。】

#### 4.4.5 使用消息传递进行同步

共享内存是一种非常强大的机制，它支持灵活的匿名通信，而且它的单芯片 CMP 实现可以减少与多芯片 SMP 中共享内存相关的许多开销。尽管如此，在多核 SMP 的核心间的消息传递仍可能存在，因为将数据传输和同步结合在一起的消息特别适合于生产者-消费者通信。【MPI 是事实上的标准，OpenMP 也常被使用。】

### 4.5 可靠性

第二部分的 CW #3 指出，下一代微处理器将面临更高的软件错误率和硬件错误率。不可靠组件得到可靠系统的一种方式是在空间或时间的冗余。由于空间冗余意味着更高的硬件成本和更高的功耗，因此我们必须在多核设计中慎重使用冗余。对任何具有唯一数据副本的内存使用单错误校正、双错误检测(SEC/DED)编码，并在任何仅具有可从其他地方检索的数据副本的内存中，使用奇偶校验来保护，而违反这些准则的服务器则会出现可靠性问题 [Hennessy and Patterson 2007]。【还有系统级可靠性，这是一个严重的问题。】

例如，如果 L1 数据高速缓存使用回写写入的 L2 高速缓存，则 L1 数据高速缓存只需要奇偶校验，而 L2 高速缓存需要 SEC / DED。SEC / DED 的成本是字宽对数的函数，对于 64 位的数据来说，8 位的 SEC / DED 的是非常合适的。奇偶校验

每个字节只需要一位。因此，能源和硬件的成本适中。

大型机是可靠硬件设计的黄金准则，而它们使用的技术之一是使用重复重传尝试克服软错误。例如，他们会在放弃并向操作系统声明发现错误之前，重试这个传输 10 次。尽管在每条总线都配备这样一种机制可能会很昂贵，但某些场景可能是经济有效的。例如，我们认为多核的通用设计框架将是全局异步的，而每个模块是本地同步的。可以通过单向链接和队列将这些较大的功能块连接在一起，而将奇偶校验和有限的重传方案纳入此框架会使得实现相对容易。

还可以将“可靠性增强”加入到包含的机制中以提高性能或简化编程。例如，上面的事务存储器（第 4.4.3 节）通过将所有内存事件回滚到事务的开始，以防错误地预测并行性来简化并行编程。这样的回滚机制可以被用来帮助解决软错误。虚拟机还可以通过在不同的虚拟机中运行不同的程序来帮助系统应对故障（参见第 6.2 节）。在硬件停止之前，虚拟机可以将应用程序从发生故障的处理器移动到多核芯片中的工作处理器。由于虚拟机提供的强大隔离性，虚拟机也可以帮助应对软件故障，从而使应用程序崩溃的可能性大大降低。

除了这些看似明显的要点之外，在多核时代的可靠性还有一些开放的问题：

- 检查错误的正确粒度是什么？整个处理器，甚至到寄存器？
- 对于一个错误，它的正确处理方式是什么？重试？还是拒绝去使用将有问题的组件？
- 错误有多严重？我们是否需要冗余线程，还是需要足够的硬件冗余？

## 4.6 性能和能耗计数器

性能计数器最初是为帮助计算机架构师评估其设计而创建的。由于它们的价值主要是内省，所以在发展过程中它们的优先级最低。鉴于这种观点和优先级，重要的性能事件的度量往往不准确或甚至是缺失的：为什么只在对产品架构师有用的性能计数器中，延缓 bug 的产生？

摩尔定律和存储墙的结合使得架构师可以设计越来越复杂的机制，并试图通过指令级并行和缓存提高性能。由于总目标是在不改变的情况下更快地运行标准程序，对于编译器编写者和程序员而言，因为他们得了解如何使他们的程序运行得更快，所以他们越来越重视性能计数器，而架构师并没有注意到这一点。因此，

这种对性能计数器的傲慢态度，使得性能计数器的责任落在了提供方，即使在连续处理器上也是如此。【芯片应该出两个版本，Debug 版本带详细的性能和能耗计算器，Run 版不带这些，这将有助于把程序开发和程序运行分开。】

提到并行编程（编译器和程序员明确负责性能），意味着性能计数器必须成为最高等级。除了监控传统的顺序处理器性能特性之外，新计数器还必须帮助解决高效并行编程的难题。

下面的 7.2 节列出了评估并行程序的效率度量标准，这表明性能计数器可以有利于多核架构获得成功：

- 为了最大限度地减少远程访问，除了传输本地访问和本地字节外，还要识别和计算远程访问的数量和移动的数据量。
- 为了平衡负载，识别和测量空闲时间与每个处理器的活动时间。
- 为了减少同步开销，识别和测量每个处理器同步花费的时间。

随着能源和能耗（power and energy）日益重要，它们也需要进行测量。从能源和能耗的角度来看，电路设计人员可以为重要模块创建焦耳计数器。在台式计算机上，主要的能耗损失是在处理器、主存储器、高速缓存、内存控制器、网络控制器和网络接口卡上。

基于焦耳和时间，我们可以计算瓦特。不幸的是，测量时间变得越来越复杂。由于时钟速率是固定的，因此处理器通常会对处理器时钟周期进行计数。为了节省能源和电力，一些处理器具有可调阈值电压和时钟频率。因此，为了准确地测量时间，除了时钟周期计数器之外，我们现在需要“皮秒计数器”。

性能和能量计数对于并行处理的成功至关重要，而且好消息是它们比较容易被包含。我们的主要观点是提高他们的优先级：如果程序员无法准确测量其影响，请不要在程序中包含对性能或能量有显著影响的功能。

## 5.0 编程模型

图 1 显示，编程模型是系统开发人员的应用程序的自然模型与可用硬件上该应用程序的实现之间的桥梁。一个编程模型必须允许编程者能够平衡高程序性能和高开发效率（**productivity and implementation efficiency**）两个具有挑战性的目标【好的应用程序性能意味着与体系结构结合得更加紧密，这意味着面向体系结构的优化。而开发效率最高的是不要改程序。**Productivity** 本来应该翻译成生产率，但是感觉不是特别。】。在并行化应用程序时，实现效率始终是一个重要目标，因为总是可以按顺序运行具有有限性能需求的程序。我们认为实现这一平衡的关键是两个相互冲突的目标：

- 对程序员的不透明性抽象了底层的体系结构。这样的抽象避免了程序员去学习体系结构中复杂的细节，并提高了程序员的生产率。
- 可见性使程序员能看到底层硬件的关键因素。它允许程序员通过探索设计参数（如线程边界，数据局部性和应用程序元素的实现）来实现应用程序的性能提升。

由于实现未来多核的性能功耗比最大化是非常重要的，这其中成功的关键就在于程序员提升程序性能的能力。

图 7 显示了目前在不透明度/可见性折衷方面缺乏同一意见。它列出了五个关键并行任务的十个编程模型示例，这些任务需要程序员做出明确的决定来实现几个任务的速度，同时兼顾运行性能。在这些极端中，程序员完成一部分任务，并将其余部分交给系统。

提高抽象层次与提升性能又是矛盾的。很低的抽象层次可能会实现性能，但代价是产生了软件产出率问题。而这已成为信息技术行业的主要障碍。过高的抽象层次也会降低产出率，因为程序员将被迫花费很多时间来理解抽象层以实现性能。

在接下来的章节中，我们将向并行机编程系统的设计者提出一些建议。并非仅仅关注硬件、应用程序或数学形式，而是创建和评估更多受心理学结果启发的编程模型（第 5.1 节）。成功的并行模型的一些看似明显但常常被忽略的特性能够在不影响效率的前提下提高抽象层次，这使得程序能够独立于处理器的数量

（第 5.2 节），并支持丰富的数据类型（第 5.3 节）以及支持并行性的风格在过去被证明是成功的（第 5.4 节）。

模型	域	任务标识	任务映射	数据分布	通信映射	同步
Real-Time Workshop [MathWorks 2004]	DSP	详细的	详细的	详细的	详细的	详细的
TejaNP [Teja 2003]	网络	详细的	详细的	详细的	详细的	详细的
YAPI [Brunel et al 2000]	DSP	详细的	详细的	详细的	详细的	不需详细的
MPI [Snir et al 1998]	HPC	详细的	详细的	详细的	不需详细的	不需详细的
Pthreads [Pthreads 2004]	通用	详细的	详细的	不需详细的	不需详细的	详细的
StreamIt [Gordon et al 2002]	DSP	详细的	不需详细的	详细的	不需详细的	不需详细的
MapReduce [Dean and Ghemawat 2004]	大数据集	详细的	不需详细的	不需详细的	不需详细的	详细的
Click to network processors [Plishker et al 2004]	网络	不需详细的	不需详细的	不需详细的	不需详细的	详细的

OpenMP [OpenMP 2006]	HPC	不需详细的 (指令, 有些是详细的)	不需详细的	不需详细的	不需详细的	不需详细的 (指令, 有些是详细的)
HPF [Koelbel et al 1993]	HPC	不需详细的	不需详细的	不需详细的 (指令)	不需详细的	不需详细的

图 7. 针对五种重要任务的十种当前并行编程模型的对比, 越需要详细的排序就靠前。高性能计算应用 (Pancake and Bergmark 1990) 和嵌入式应用 (Shah et al 2004a) 建议这些任务必须由编程模型统一成某一种路线: 1) 将应用程序划分为并行任务; 2) 将计算任务映射到处理元素; 3) 数据分配到存储元件; 4) 将通信映射到互连网络; 5) 任务间同步。

## 5.1 心理学研究启发下的编程模型尝试

在保证并行应用的高产出和高效率实现的前提下, 开发编程模型是未来众核系统部署面对的最大挑战。因此, 编程模型的研究是一个高度优先事项。我们认为, 过去的编程模型开发以硬件为中心、以应用程序为中心或者以形式为中心的。硬件制造商自己通常开发以硬件为中心的编程模型, 试图最大限度地提高硬件的生产效率。例如, 被称为 IXP-C [Intel 2004] 的 C-变体以及被称为微块的库元素都是为 Intel IXP 系列网络处理器而开发的[Adiletta et al 2002]。这样的环境并不能提供一个令人满意的产出率提升, 也不会支持更广泛的并行编程过程——体系结构设计、调试等——涉及并行应用程序的开发。

以应用程序为中心的编程模型 (如 Matlab [MathWorks 2006]) 通常侧重于简化相关应用程序域的开发。这些模型也不支持更广泛的并行编程过程, 也不支持微调实现来实现效率约束。

以形式为中心的编程模型, 如 Actors [Hewitt et al 1973], 通过使用简洁的句法减少程序员犯错误的机会, 并通过验证代码部分的正确性来提供删除错误的机

会。

所有三个目标显然很重要：效率，生产力和正确性。然而，令人惊讶的是，心理学研究几乎没有影响，尽管这些模型的成功将受到使用它们的人的强烈影响。来自心理学研究团体的测试方法已被用于 HCI，但在语言设计和软件工程方面极其缺乏。例如，有一个调查人类错误的原因的丰富理论，这在人机交互领域中是众所周知的，但显然它没有渗透到编程模型和语言设计领域。[Kantowitz 和 Sorkin 1983] [原因 1990]已经有一些初步尝试来确定软件工程（SE）和人机交互（HCI）领域之间合作的系统性障碍，并提出对 CS 课程进行必要的改变以使这些领域一致。迄今为止这些提案迄今未取得实质性进展。[Seffah 2003] [Pyla et al 2004]我们认为，将人类心理学和解决问题的研究集成到广泛的设计、编程、调试和维护复杂并行系统的广泛问题中，对于开发广泛成功的并行编程模型和环境至关重要。

事务内存是编程模型的一个例子，有助于防止人为错误。程序员很难确定何时在并行代码中进行同步，并且经常出错。事务性内存的一个优点是，即使程序员对并行化代码的安全性做出了不正确的假设，系统也能确保正确性。事务内存的目的并不主要是效率、形式或甚至是产出率；它的主要目的是当程序员犯错的时候或者整个自动并行编译器出错的时候仍然能够保证正确地运行。

我们不仅忽略了我们在编程模型设计中对人类认知的见解，我们也没有遵循他们的实验方法来解决人们如何使用它们的争议。这种方法是人体实验，这种实验非常普遍，大多数校园都有委员会，在进行这些实验之前必须咨询委员会。将我们关于过程或编程的假设置于正式测试之下往往会产生意想不到的结果，这直接影响我们的直觉 [Mattson 1999]。

一个小例子是关于使用共享内存的编程和消息传递的对比研究。这些替代方案几十年来一直是热门辩论的主题，而且哪个更好，什么时候哪个更好尚未取得统一认识。最近的一篇文章比较了新手程序员为小型并行处理器编写的小程序的效率和产出率[Hochstein 等人，2005]。尽管这不是辩论的最后一句话，但它确实表明了尝试解决重要编程问题的途径。幸运的是，越来越多的用户研究团队尝试评估计算机语言的生产力[Kuo 等人，2005] [Solar-Lezama 等人，2005] [Ebcioğlu 等人，2006]。



我们相信未来成功的编程模型必须更加以人为本。它们将针对人工流程进行量身定制，在同等复杂的众核硬件上高效地实施，调试和维护复杂的并行应用程序。此外，我们必须使用以人为主体的实验来为我们解决开放性问题，并就如何真正简单并高效地在众核系统上编程这一问题上取得进步。

【以心理学为基础的想法，尽管可能难以实现，但是确实很有意思。程序 Bug 会严重影响程序员的心理健康，应该属于职业病的一种吧。】

## 5.2 型号应与处理器的数量无关

MPI 是当前并行科学编程的主要编程模型，它迫使编码人员意识到计算任务到处理器的精确映射。这种方式多年来一直被认可，它增加了程序员的认知负担，一直使用它是因为它易于表达并且能够提供最佳性能 [Snir 等人 1998] [Gursoy 和 Kale 2004] 。

因为我们预计可利用的并发性会大量增加，所以我们认为这个模型在不久的将来会崩溃，因为程序员必须明确地处理分解数据，映射任务以及在数千个处理元素上执行同步。

最近在编程语言方面的发展都集中在这个问题上，他们的产品提供了处理器数量未知的模型。[Deitz 2005] [Allen et al 2006] [Callahan et al 2004] [Charles et al 2005]。虽然这很有吸引力，但是这些模型具有一个问题--提供性能。在许多情况下，可以提供提示来共同定位特定存储器域中的数据和计算。此外，因为程序没有过多规定，所以系统在映射和调度方面具有相当大的自由度，理论上可以用来优化性能。然而，这一协议的交付使用仍然是一个开放的研究问题。

【MPI 是高性能并行计算事实上的编程标准。节点间用 MPI，节点内采用其他细粒度数据级并行是一个可行的方法。MPI 的抽象比较好，而且是分布式内存的，这样能够极大化地利用串行程序的计算能力，其他编程难以实现。】

## 5.3 模型必须支持各种各样的数据大小和类型

尽管第 3 节中的算法在嵌入式和服务器基准测试中通常是相同的，但对于数据类型并非如此。PEC 依赖于单精度和双精度浮点数和大整数数据，而 EEMBC 使用从 1 到 32 位变化的整数和定点数据。[EEMBC 2006] [SPEC 2006]请注意，大

多数编程语言仅支持 40 年前的 IBM 360 中最初发布的数据类型的子集：8 位字符，16 位和 32 位整数以及 32 位和 64 位浮点数。【应该再吹一波 IBM。】

这导致了相对明显的现象。如果并行研究事项启用了新语言和编译器，它们应该允许程序员至少指定以下长度（和类型）：

- 1 位（布尔）
- 8 位（整数，ASCII）
- 16 位（整数，DSP 定点，Unicode）
- 32 位（整数，单精度浮点，Unicode）
- 64 位（整数，双精度浮点数）
- 128 位（整数，四精度浮点数）
- 大整数（> 128 位）（加密）【加密解密，如 RSA 密码分析等。】

BLAS 例程已经开始出现用于输入，内部计算和输出的混合精度浮点精度 [Demmel 等 2002]。需要一个类似的，也许更灵活的结构，以便所有方法都可以利用它。虽然所有这些类型的支持主要可以完全用软件提供，但我们不排除额外的硬件来协助实现更加广泛的数据类型兼容。

除上述更“原始”的数据类型外，编程环境还应提供分布式数据类型。这些与其并行性风格自然紧密耦合，因而会影响整个设计。DARPA 目前正试图使用高效语言系统生产计划中发布的语言来解决这个问题，主要关注的是支持用户指定的发行版。

## 5.4 模型必须支持已有的并行性风格

编程语言，编译器和体系结构往往把注意力放在一种并行编程风格上，通常迫使程序员用这种风格表达所有的并行性。现在我们已经进行了几十年的这种实验，我们认为结论很清楚：某些类型的并行性已经在某些应用中证明是成功的，并且没有任何一种风格被证明是最好的。

我们认为编程模型和架构不应该把所有的鸡蛋放在一个篮子里，而应该支持各种样式，这样程序员才能在使用时进行更好地选择。我们认为该清单至少包括以下内容：

1. 独立任务并行性是一种易于使用的正交并行方式，应在任何新架构中支

持。老式矢量计算机作为一个反例，尽管有许多并行的功能单元，但却无法利用任务级别的并行性。事实上，这是向量计算机向大规模并行处理器转换时使用的主要论点之一。【随着摩尔定律的消失，向量化单元将越来越重要，这是提升程序运行性能的一种卓有成效的方式。】

2. 字级并行是与一些低层次（如稀疏和稠密的线性代数和结构化网格）的自然匹配。它支持的例子包括编程语言中的数组操作，矢量化编译器和矢量架构。向量编译器会在编译时提示为什么一个循环没有进行向量化，非计算机科学家可以通过向量化代码，因为他们理解并行模型。多年以来，它可以说是一种新的并行语言，编译器和体系结构。

3. 比特级并行可以在使得处理器内在功率，面积和时间面积和时间上比在处理器之间更高效地被利用。例如，用于密码学的安全散列算法（SHA）具有明显的并行性，但是其形式要求在小型领域上的操作之间需要非常低的等待时间通信。

除了并行性的风格之外，我们也有内存模型的问题。因为并行系统通常包含分布在整个机器中的内存，所以问题往往出现在程序员对这个内存的认识上。提供统一地址的虚拟系统在程序员中非常流行，但是，将这些扩展到大型系统仍然是一个挑战。当多个处理器可以更新相同的位置时，内存一致性问题（与本地和远程内存操作的可见性和排序有关）就会出现，每个位置可能都有缓存。显式分区的系统（如 MPI）避开了类似的问题，但程序员必须处理自己执行远程更新的低级细节。【分布式内存一致性问题应该是一个很麻烦的事情。】

## 6.0 系统软件

除编程模型外，编译器和操作系统还有助于跨越图 1 中应用程序和硬件塔之间的差距。我们认为，这些重要计划在过去几十年中已经发展得如此之大，以至于在我们转向并行时很难做到可能需要的创新。因此，我们建议更多地依靠搜索来产生高效的并行代码的自动调优（6.1 节），而不是完全重新设计并行编译器。我们建议更多地依靠虚拟机和系统库来包含那些仅仅被应用需求的功能（参见第 6.2 节），而不是依赖传统的大型单片机操作系统。

### 6.1 自动调优与传统编译器的比较

不管编程模型如何，未来并行应用程序的性能关键取决于所生成代码的质量，这在传统上属于编译器的职责。例如，它可能需要选择适当的同步结构实现或优化通信语句。此外，编译器必须生成良好的顺序代码；复杂的微架构和内存层次结构的任务。编译器选择要执行的优化，为这些优化选择参数，并从库内核的其他实现中选择。优化选择的结果空间很大。这样的编译器将从程序中隐式或显式显示的并行性开始，并尝试增加其数量或修改它的粒度——这是一个可以简化的问题，但并不是通过一个好的编程模型来避免的。

#### 6.1.1 提高现代编译器的难度

不幸的是，很难向编译器添加新的优化，这可能需要从指令级并行转换到任务级和数据级并行。作为一个现代的编译器，包含了数百万行代码，而新的优化通常需要对其内部数据结构进行根本的更改。因此，大型的设计投资难以实现，因为与语言标准的兼容性和生成代码的功能正确性要远优先于输出代码质量。此外，对于所有可能的输入和在研究会议上发表论文所要求的少数测试用例来说，外来的自动优化传递是困难的。因此，用户已经习惯了关闭复杂的优化，因为它们会触发更多的编译程序错误。【高级别的优化需要高级别的抽象，这一点目前编译器难以达到，不知道深度学习是否可以。】

由于现有编译器的局限性，峰值性能可能仍然需要用 C、FORTRAN 或甚至汇

编代码等语言手动编写程序。实际上，大多数可伸拓展并行代码都有数据布局、数据移动和处理器。由程序员手动编写的处理器同步。这种低级编码是劳动密集型的，通常不能移植到不同的硬件平台，甚至不能移植到后期相同的指令集架构实现。【是的，面向体系结构的优化很难在跨平台，因为与具体体系结构绑定了。】

### 6.1.2 基于搜索的自动调优调优的前景

我们的设想是，依靠嵌入各种软件合成的搜索可以解决这些问题。通过搜索合成高效的程序已经在代码生成的几个领域中得到了应用，并且取得了一些显著的成功 [Massalin 1987] [Granlund et al 2006] [Warren 2006]。

近年来，“自动调优” [Bilmes et al 1997] [Frigo and Johnson 1998] [Frigo and Johnson 2005] [Granlund et al 2006] [Im et al 2005] [Whaley and Dongarra 1998] 作为生产高质量便携式科学代码的有效方法而受到欢迎。通过生成许多给定内核的变量，并在目标平台上对每一个变量进行基准测试，从而优化了一组库内核。搜索过程有效地尝试了许多或所有优化开关，因此可能需要花费数小时才能在目标平台完成整个搜索过程。但是，在安装库时，只需要执行一次搜索。生成的代码通常比简单的实现快好几倍，并且可以使用单个自动调优为各种机器生成高质量代码。在许多情况下，自动调优的代码比专门为目标机器手动调整的商业库更快！这种令人惊讶的结果部分地解释了自动调优调优不知疲倦地尝试许多特殊程序的许多不寻常变量的方式，通常会找到导致更好性能的非直观循环展开或注册阻塞因素。

例如，图 8 显示了 Itanium2 上性能在方块选择中如何在 4 倍内变化。从自动调优的经验可以看出，通过搜索许多优化参数的可能组合，我们可以避开为优化策略创建有效启发式的问题。

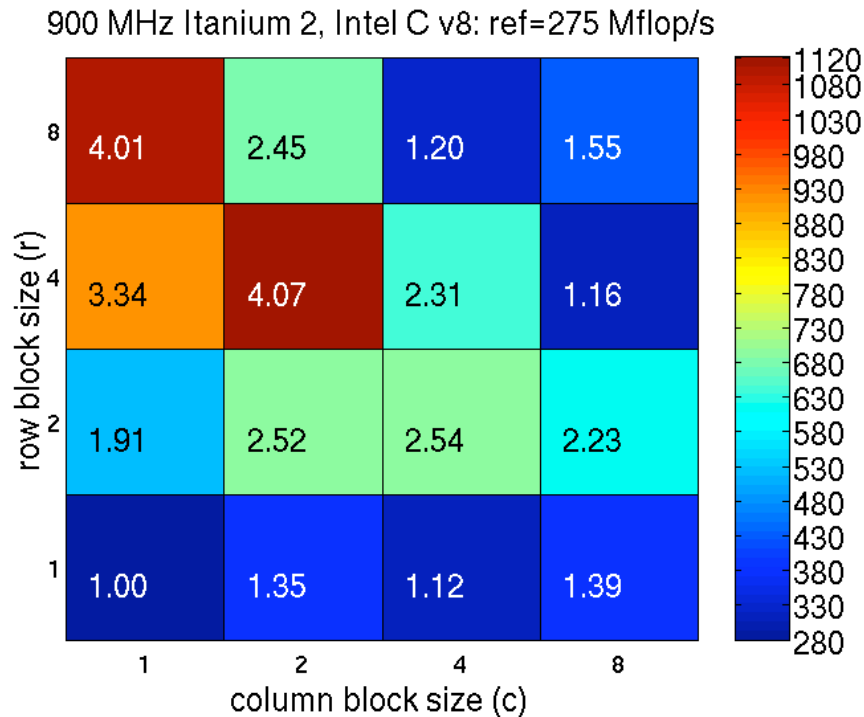


图 8. 在使用分块压缩稀疏行 (BCSR) 格式的有限元问题上, Itanium 2 上的稀疏矩阵性能[Im et al 2005]。针对所有区分 8x8-16 实现的区块大小显示性能(颜色编码, 相对于 1x1 基线)。这些实现完全展开最内层的循环, 并使用源和目标向量的标量替换。随着 r 和 c 的增加, 您可能会合理地预期性能会相对平稳地增长, 但显然并非如此。平台: 900 MHz Itanium-2, 3.6 Gflop / s 峰值速度, 英特尔 v8.0 编译器。

自动调优的普及可能导致基准测试的变化。常规的基准测试, 如 SPEC, 是作为源代码发布的, 必须原封不动的进行编译和运行。此代码通常包含支持特定目标计算机的手动优化, 例如特定的缓存阻塞。然而, 自动调优的代码将允许基准自动为每个目标找到最佳的方法。

### 6.1.3 将自动调优扩展到并行

我们相信自动调优也可以帮助编译并行代码。然而, 并行架构引入了许多新的优化参数, 到目前为止, 还没有成功的并行代码自动调优。对于任何给定的问题, 可能有几个并行算法, 每个算法都有可替代的并行数据布局。最优选择不仅



取决于处理器架构，还取决于计算机的并行性，以及网络带宽和延迟。因此，在并行设置中，搜索空间可能比顺序内核大得多。【[结点间并行，结点内优化，算法优化。](#)】

为了减少搜索空间，可以将搜索的良好数据布局和通信模式与搜索好的计算内核分离开来，特别是在使用性能模型的过程中。网络和内存性能可以相对较快地使用测试模式进行描述，然后插入到网络的性能模型中，从而为搜索计算内核提供合适的代码循环。【[性能优化对于程序员而言是非常繁琐的事情](#)】

## 6.2 解构操作系统的支持

虽然程序模型居于操作系统层之上，但是该操作系统层的效率很大程度上会影响到那些依赖于它的程序的效率。正如处理器越过了益处变大的拐点，我们相信操作系统也是如此。展望未来，我们相信操作系统必须被解构，虚拟机使终端应用程序能够仅选择所需的操作系统功能的一部分，而不是被迫接受庞大的软件栈。正如硬件正从单一的单片处理器转移，操作系统可能正在从一个单一的程序转移。我们在这一部分阐述了这些论点。

### 6.2.1 嵌入式计算中对保护的增长需求

过去操作系统最矛盾的地方在于它的嵌入式操作系统和服务器操作系统之间。嵌入式系统历来具有非常少的专用运行时系统，对实时调度进行严格控制，但几乎不支持保护和虚拟化。这反映了降低处理器成本需求，内存占用和功耗的愿望以及制造商为特定嵌入式系统定制软件的假设。传统的服务器操作系统拥有数百万行代码，并提供了非常丰富的功能。保护和虚拟化对于支持使用一系列写入行业标准 API 的第三方代码构建的大型软件系统以及通过不安全的全球互联网进行通信至关重要。

随着嵌入式系统功能的增加，我们相信这两个世界正在发生碰撞和合并。例如，手机和游戏机现在支持数千兆字节的文件系统和复杂的 Web 浏览器。特别是，由于可靠性问题，先前拒绝安装第三方软件的手机制造商现在认识到必须提供标准 API 以允许用户扩展，这将需要更复杂和稳定的操作系统和硬件支持这些

需求。

由于嵌入式计算机越来越多地连接到网络，我们认为它们将越来越容易受到病毒和其他攻击的攻击。实际上，第一台个人电脑操作系统因为开发人员认为个人电脑只有一个用户而失去了保护，直到我们将个人电脑连接到互联网后，这些用户才能正常工作。想象一下，如果在他们加入互联网之前安全性已经成为 PC 操作系统的优先考虑，我们的生活会变得多好。

【安全是永恒的需求】

## 6.2.2 虚拟机的援助

虽然传统的操作系统太大而且脆弱难以支持快速的创新，但包含数百万条对应用程序功能至关重要的重要遗留代码。重新对虚拟机（VMs）产生兴趣表明操作系统已经达到了自己的技术拐点。虚拟机技术允许将具有正在运行的应用程序的完整操作系统视为由虚拟机监视器（VMM）或虚拟机管理程序操纵的软件组件。VMM 在客户操作系统和硬件之间插入一个精简的软件层，以便给客户操作系统一个错觉，即它正在自己的真实硬件副本上运行。这种方法允许一个非常小的，开销很低的 VMM 提供创新的保护和资源共享，而无需运行或修改数百万行的操作系统。

虚拟机似乎是服务器操作系统的未来。例如，AMD，Intel 和 Sun 都修改了指令集体系结构以支持虚拟机。虚拟机在服务器计算中很流行，原因如下：

- 提供更高程度的病毒和攻击的防范；
- 通过隔离单个虚拟机内的程序来应对软件故障，以便于不损害其他程序；
- 在不停止程序的情况下，通过将虚拟机从一台计算机迁移到另一台计算机应对硬件故障。

VMM 支持的这些功能为常规操作系统的故障提供了一个优雅的解决方案。VMM 与众核系统也非常匹配，因为在 1000 个处理器上运行多个应用程序时，空间共享将变得越来越重要。

VMM 的成本是多少？在 VMM 上运行操作系统的开销通常是指令集体系结构的功能。我们相信嵌入式和服务器的许多核心架构应支持虚拟化，因为硬件成本微不足道。通过将指令集体系结构设计为可虚拟化，软件开销可能非常低。事



实上，一个重要的体系结构目标是提供有助于防止 VMM 随时间扩展的支持。

【虚拟化是一个重要的方向】

### 6.2.3 解构操作系统

Rosenblum 认为，在提供保护和共享硬件资源的精简 VMM 层之上，服务器操作系统的未来本质上可以是只有需要的功能链接到应用程序的库。[Rosenblum 2006]今天，这个愿景与嵌入式操作系统类似。例如，VxWorks 允许用户选择 OS 的哪些功能将包含在此嵌入式应用程序中。[Wind River 2006]因此，我们看到操作系统在嵌入式和服务端计算方面有更多共同点。

虽然这个愿景很有吸引力，但它并不具有约束力。应用程序可以在 VMM 之上运行非常薄或非常厚的操作系统，甚至可以同时运行多个操作系统以适应不同的任务需求。例如，运行在不同内核上的实时代码和尽力而为代码，或者多个高密度内核上的最小数据平面操作系统以及大型通用内核上的复杂控制平面操作系统。【另一个优化层次】

## 7.0 成功的衡量标准

在图 1 的完整桥上讨论了六个问题，我们需要决定怎样更好地发现和评估这些问题的答案。在接下来的研究中，我们关注最大化两个指标——程序员的生产力和最终的实现效率——并提供一个工具来帮助研究人员更快地创新。【就是系统评测，这个课程主要关心的东西。】

### 7.1 最大化程序员生产力【开发效率】

单个芯片上的数千个处理元素对应用程序设计人员来说是一个很大的编程挑战。由于在这些设备上正确和有效地实现应用程序很困难，所以当前的片上多处理器的应用一直很缓慢。例如，业界媒体报道称目前用于网络应用的片上多处理器称(2004 年 Weinberg):

“.....有强大和复杂 packet-engine 集的网络处理器已被证明是难以编程的。”

早期的片上多处理器失败，如 TI TMS320C80，完全是因为应用程序设计者不能高效地利用他们的表现。因此，能够在这些高性能多处理器上高效编程的能力至少与提供这些架构的高性能芯片实现一样重要。

另一个值得考虑的领域是辅助语言生产力特性的硬件结构的添加。例如，在完全用软件支持事务内存可能太慢而无法使用，但是可以通过硬件支持来提高效率。其他的例子包括支持垃圾收集、细粒度同步(Cray MTA)、单向消息传递、调试的跟踪收集[Xu et al 2003]，以及性能和能量计数器来帮助程序优化(参见第 4.5 节)。

生产力是一个多方面的术语，难以量化。然而，案例研究如[Shah et al 2004b]和正在进行的 DARPA HPCS 项目[HPCS 2006]的工作建立了我们的信心，即生产力是可以进行定量比较的。此外，在编程心理学方面的工作也可以告知我们的评估工作。【开发效率除了与人、编程模型、编程语言相关外，与算法库、SDK 的完备性也是相关的。】

## 7.2 最大化应用程序性能

图 2 的一个含义是，在 15 年的时间里，应用程序的性能稳定地增加，仅仅是通过在新一代的处理器上运行应用程序，而很少增加程序员的工作量。随着处理器性能增长的放缓，需要新的想法来进一步实现应用程序性能的提升。由于现有的文献只显示了小部分的应用领域的成功，例如用于网络应用程序的 Cisco 的 188 处理器的 Metro 芯片[Eatherton 2005]，因此需要激进的想法才能使众核架构成为高效软件开发的安全和可靠基础。

此外，能耗限制使我们不得不承认单个处理单元的最大性能的极限，因此我们必须通过优化总系统性能来赢得应用效率的提升。这将需要额外的设计空间探索。[Gries 2004]对设计空间探索的一般文献进行了广泛的回顾，并在[Gries and Keutzer 2005]中提出了利用 CoWare 或 ten 二氧化硅工具进行嵌入式处理器设计空间探索的商业软件支持。然而，评估完整的应用程序需要的不仅仅是机敏的处理元素定义；必须探讨完整的系统架构设计空间，包括内存和互联。虽然这些设计空间探索的重点是嵌入式处理器，但我们相信，众核系统的处理器看起来更像嵌入式处理器，而不是当前的桌面处理器(参见 4.1.2.)。

新的效率指标将构成对新的并行体系结构的评价。正如在串行世界中一样，从程序执行中有许多“可观察的”，提供了一些提示(如缓存缺失)到运行程序的整体效率。除了串行性能问题之外，并行系统体系结构的评估将重点放在：

- 最小化远程访问。在通过不同处理元素分布的计算任务访问数据的情况下，我们需要优化它的位置，以便使通信最小化。
- 负载平衡。对处理元素的计算任务的映射必须以尽可能少的空闲(等待数据或同步)的方式执行。
- 数据移动和同步的粒度。大多数现代网络在大数据传输方面表现最佳。

此外，同步的延迟也很高，因此尽可能少地进行同步是有利的。

嵌入式系统的软件设计环境，如[Rowen 和 Leibson 2005]中所描述的，提供了更大的支持来实现这些类型的系统级决策。为了帮助程序员实现这些目标，我们推荐可以测量这些性能问题的硬件计数器(参见第 4.6 节)。

过去十年探索新架构的传统途径是模拟。我们怀疑软件模拟本身会提供足够的吞吐量，以便对许多系统体系结构进行全面评估。对于需要较长开发周期的项

目硬件原型也不足以满足要求。这些临时原型的开发将非常缓慢，无法影响行业在未来多核体系架构上需要做出的决策。我们需要一个平台，在这个平台上，软件实验反馈的新型多核体系架构运行具有代表性工作负载的实时应用程序，将在几天内而不是几年内形成新的系统体系结构。

## 7.3 RAMP：多处理器的研究加速器

多处理器研究加速器（RAMP）项目是六个机构中 10 位教职工的开源工作，旨在创建一个计算平台，以实现并行软件和体系结构的快速创新[Arvind et al 2005] [Wawrzynek et al 2006]。RAMP 的灵感来自：

1. 研究人员难以建立现代芯片，如第 2 节 CW#5 所述。
2. 现场可编程门阵列（FPGA）的快速发展，其容量每 18 个月翻一番。FPGA 现在具有数百万门和数百万位内存的容量，并且可以像修改软件一样轻松地重新配置 FPGA。
3. 只要性能足够快，以及时进行实验，灵活性，大规模和低成本就能为研究人员带来绝对的性能。这个观点表明使用 FPGA 来进行系统仿真。
4. 更小更好（见 4.1 节）意味着当今许多这些硬件模块都可以安装在 FPGA 内部，避免了单个模块必须跨越许多 FPGA 时过去更严格的映射问题。
5. 来自 Opencores.org，Open SPARC 和 Power.org 的开放源代码模块的可用性，可以毫不费力地插入 FPGA [Opencores 2006] [OpenSPARC 2006] [Power.org 2006]。

虽然 RAMP 的想法只有 18 个月的时间，但该组织已经取得了快速的进展。它拥有 NSF 和几家公司的财务支持，并且有基于老一代 FPGA 芯片的硬件。尽管 RAMP 的运行速度比实际硬件慢 20 倍，但它会模拟许多不同速度的组件，以准确地报告在模拟时钟频率下的正确性能。

该小组计划开发三个版本的 RAMP 来演示可以做什么：

- 集群 RAMP（“RAMP Blue”）：由 Berkeley 团队领导，该版本将使用 MPI 进行高性能应用的大规模示例，例如 NAS 并行基准（NPB）[Van der Wijngaart 2002]或 TCP / IP，用于 Internet 搜索。8 板版本将在 256 个处理器上运行 NPB。

- 事务性内存 RAMP (“RAMP Red”)：由斯坦福大学领导，该版本将使用事务性内存的 TCC 版本实现高速缓存一致性[Hammond et al 2004]。单板系统比事务内存模拟器运行速度快 100 倍。
- •Cache-Coherent RAMP (“RAMP White”)：由 CMU 和德克萨斯特遣队领导，该版本将实现基于环的一致性或基于窥探的一致性。

所有人都将共享相同的“gateway”——处理器，内存控制器，交换机等 - 以及 CAD 工具，包括协同仿真。[Chung 等 2006]

目标是在网站上免费提供“gateway”和软件，重新设计电路板以使用最近推出的 Virtex 5 FPGA，最后找到一家制造商以低利润销售它们。估计每个处理器的成本约为 100 美元，每个处理器的功耗约为 1 瓦，产生一个价值约 10 万美元的 1000 个处理器系统，消耗大约 1 千瓦，并且占据标准机架空间的大约四分之一。

大规模并行编程模型、实时约束、保护和虚拟化之间的交互为架构和软件系统研究提供了丰富的基础。希望大规模多处理、标准指令集和操作系统的优点，包括低成本、低功耗和易变性将使 RAMP 成为许多类型研究人员并行研究的标准平台。如果它在将许多学科整合在一起时产生了“水坑效应”，它可能会导致创新，从而更快速地为图 1 中的七个问题找到成功的答案。

## 8.0 结论

第二部 CWs#1、7、8 和 9 的传统观点表明：能耗、内存和指令级并行墙三重冲击迫使微处理器制造商在并行微处理器赌上他们的未来。然而这也并非把握十足，因为并行软件具有不平衡路线的记录。

但从研究的角度来看，这又是一个令人兴奋的机遇。实际上，任何改变，如果他能保证更容易地写出在多核计算系统上高效执行的程序，那么这些改变就都是合理的，例如：新的编程语言、新的指令集架构、新的互联协议等等。

这个机遇激发了我们这个具有不同背景，却在伯克利齐聚一堂的团队，花费近两年来讨论这个课题，从而产生了图 1 的七个问题及以下非传统观点：

- 关于多核 multicore 与众核 manycore：我们认为 manycore 是计算的未来。此外，假设适用于 2 到 32 个处理器的 multicore 架构和编程模型可以逐步演化为服务 1000 个处理器的 manycore 系统是不明智的。【看看 GPU，确实如此。但是 GPU 等众核架构性能受限。】“适用于 2 到 32 个处理器的 multicore 架构和编程模型可以逐步演化为服务 1000 个处理器的 manycore 系统”的假设是不明智的。【CUDA 和 OpenCL 编程模型与传统的 OpenMP 和多线程完全不一样。】
- 关于应用程序塔：因为我们需要研究并行编程模型及其架构，我们相信使用 13 个小矮人作为未来并行应用程序的替代品这个方法会更有前景。
- 关于硬件塔：我们建议使用简单的处理器，以便在内存和处理器设计上进行创新，并且考虑独立的延迟和面向带宽的网络。由于点对点通信模式非常少，因此我们使用根据应用需求量身定制互联拓扑结构的混合互联设计的电路交换机，其面积和功率效率可能高于全交叉开关，并且比静态网状拓扑更具计算效率。传统的缓存一致性不足以协调 1000 个内核的活动，因此我们推出更强大的硬件以支持细粒度同步和通信构造。最后，如果不能提供让程序员准确衡量其影响的计数器，我们不建议加入对性能或功耗有显著影响的功能。
- 关于桥接两个塔的编程模型：为了提高生产率，编程模型必须更加以人为中心，并解决在 manycore 硬件上开发并行应用程序相关的全部问题。为了最大限度地提高应用程序的效率以及程序员的工作效率，编程模型应该独立于

处理器的数量，它们应该允许程序员使用更丰富的数据类型和大小集，并且支持成功并且有名的并行模型：独立任务级、字级和位级并行模型。

- 我们认为自动翻译应该在编译器翻译并程序方面发挥更大或者至少是补充的作用。此外，我们同样认为传统操作系统将被解构，并且操作系统功能将使用虚拟机进行编排。
- 为了快速提供有效的并行计算路线图，以便业界可以放手一搏，我们鼓励研究人员使用自动调优和 RAMP 来快速探索这个空间，并通过测试在 manycore 系统上编程 13 个小矮人高效运行的容易程度来衡量成功与否。
- 尽管嵌入式和服务端计算在历史上就沿着不同的道路发展，但我们认为来自 manycore 的挑战使它们更紧密。通过利用其中他们发展过程中涌现出的好的想法，我们相信会找到更好的答案来解决图 1 中的七个问题。

作为试验这些观察结果有用性的案例，其中一位作者受邀参加研讨会，并被提出如果你拥有无限的内存带宽，你会做什么的问题？我们用小矮人来解决问题，找寻哪些受限于计算，哪些受限于存储。下面的图 9 给出了我们快速研究的结果，即内存延迟比内存带宽更大的问题，并且一些小矮人不受内存带宽或延迟的限制。无论我们的答案是否正确，建立一个有原则的框架来试图回答这些开放和困难的问题总是令人兴奋的。

本报告旨在为这些观点抛砖引玉。有这样一个开放，动心和紧迫的研究议程来理清图 1 的两座塔代表的概念。我们欢迎您访问 [view.eecs.berkeley.edu](http://view.eecs.berkeley.edu) 来参与到这一重要的讨论。

小矮人性能限制：内存带宽、内存延迟还是计算？

小矮人	性能限制：内存带宽、内存延迟还是计算？
1、稠密矩阵	计算受限
2、稀疏矩阵	一般 50%计算受限，50%内存带宽受限
3、频谱（FFT）	内存延迟受限
4、多体问题	计算受限
5、结构网格	目前更多是内存带宽受限
6、非结构网格	内存延迟受限

7、映射规约	具体问题具体分析
8、组合逻辑	循环冗余码问题内存带宽，密码问题计算受限
9、图遍历	内存延迟受限
10、动态程序设计	内存延迟受限
11、回溯和分支限界	？
12、建构图形化模型	？
13、有限状态机	无任何帮助

图 9 在 IBM 的建议下，封装技术可以提供几乎无限的内存带宽，这也限制了小矮人的表现。虽然内存墙限制了几乎一半的小矮人的性能，但内存延迟是比内存带宽问题更大。



## 致谢

在撰写本文期间，Krste Asanovic 利用在麻省理工学院休假期间访问加州伯克利大学。我们要感谢参加这些会议的以下人员：Jim Demmel, Jike Chong, Armando Fox, Joe Hellerstein, Mike Jordan, Dan Klein, Bill Kramer, Rose Liu, Lenny Oliker, Heidi Pan 和 John Wawrzynek。我们还要感谢那些对改进这份报告的第一稿提供反馈意见的人：Shekhar Borkar, Yen-Kuang Chen, David Chinnery, Carole Dulong, James Demmel, Srinivas Devadas, Armando Fox, Ricardo Gonzalez, Jim Grey, Mark Horowitz, Wen-Mei Hwu, Anthony Joseph, Christos Kozyrakis, Jim Larus, Sharad Malik, Grant Martin, Tim Mattson, Heinrich Meyr, Greg Morrisett, Shubhu Mukherjee, Chris Rowen 和 David Wood。根据他们广泛的评论修改报告意味着最终的草案仍需 4 个月完成！

在国防科技大学上这门课《面向高性能计算的性能评测与优化技术》期间，各位上课同学齐心协力完成翻译和校对工作，体现出良好的学术素养、技术水平和团队协作能力！《面向高性能计算的性能评测与优化技术》这门课题由刘杰教授主讲，龚春叶和甘新标博士辅助，主要讲授关于天河高性能计算机系统相关的性能评测和优化技术。

## 附加评论:

【一个研究工作是否好或者一般，应该由时间来判断，也就是所谓的让子弹先飞一会。回过头来看这篇论文的研究，尽管是在 12 年前的工作，但是其总结、分析和预测，在目前看来仍然不过时。所以时间是最好的尺度。国内科研工作，包括学术类的科学研究和技术的研发，的评价也应该由时间来评判，一个研究放到 10 后再评价可能会更加何公平公正一些。即时的学术或者技术评价陷入了数量的判别，娱乐没有什么判别，直播打赏类似即刻回报。】

【dwarf 的翻译是侏儒、矮人，这两个翻译词语在中文意境中并不友好。根据《白雪公主与七个小矮人》的故事，dwarf 翻译成小矮人更加合适。小矮人尽管矮，但是他们聪明勇敢善良。所以这篇论文标题改成了“并行计算与 13 个小矮人”，前传叫“并行计算与 7 个小矮人”。】

【回到论文本身，研究涉及应用、硬件、编程模型、系统软件和评测几个方面的内容。这些内容在目前看来仍然有借鉴的价值。应用 13 个小矮人的分类非常有意思，Intel 的图 5 也充满了真知灼见！尤其是最近火热的机器学习也有涉及，除了深度学习没有提及之外，应该还是比较完备的总结。】

【对于系统评测和编程模型，主要关注两个方面，一个是程序开发效率，一个是应用性能。这确实刻画了并行计算领域两个最重要的方面。程序开发效率非常重要。尤其经历了天河二号升级系统及 E 级验证系统相关并行应用开发之后。就一直在思考为什么 GPU/CUDA 在除了在图像处理、深度学习和虚拟货币等几个小方面取得成功之外，在高性能计算领域的成绩并不乐观。应用软件开发效率和实际应用性能是限制 GPU 大规模使用的关键。因为移植或者优化一个大型应用软件到 GPU 上，是一件非常困难的事情，需要开发细粒度数据级并行，同时要考虑访存对齐和基于可编程 Cache 的访存优化。在 CUDA 上开发和优化简单的计算核心是可行的，但是实现数万行代码的移植，目前看来挑战还是非常大。CUDA 本身是一个紧耦合的编程模型，对于异构系统，采用松散耦合的分布式方式可能是更加合适的一种方式。这种松散耦合的方式类似于 MPI 模式，也可以借鉴原有的 MPI 算法，提高软件开发效率。GPU 应用在研究上可以获利上百倍的加速效果，这种效果实际上是要打折扣的。合适的比较方式应该是新款 GPU 的最优

运行性能与最新款多核 CPU 上的最优运行性能相比。根据我们的经验，一般只有 2 倍-5 倍的加速效果。同时这种 GPU 移植和优化，带来新的问题是软件开发新功能异常困难，应用专业从业人员写串行代码就已经很困难，能在 MPI 上修改已经是非常厉害，基本上看不懂 GPU 代码，所以软件开发的可持续性非常差。Intel 的 MIC/KNL 已经停产，这个软件优化是否能够持续发生作用仍然存疑。GPU 火主要有三个方面的因素。一是 07 年左右开始的通用计算，很火热但是市场没有做起来。二是深度学习，图片训练什么的。三是挖坑，数字货币。GPU 在通用计算，尤其是科学与工程计算，是否能够发挥出宣传的功力仍然不得而知。】

【现代科技的发展存在非常大的挑战。这种挑战主要在于新知识新技术的非共享性。简单地说，在 IT 领域，最新的技术掌握在支配型公司的手里，如芯片设计的 Intel、高通，操作系统的微软、搜索和人工智能的 Google 等。这些大公司垄断了科技的发展方向。而学校，本来应该是高技术的发源地之一，越来越落后于大公司的发展。学生所接受的知识也是落后的一个时代的技术。从生意或者挣钱的角度，这样做本身没有什么问题。但是从人类社会发展的角度，科技的垄断将导致社会发展的减速。】

【这篇论文如果说有一点不足的话。那么对应用的探讨应该更加深入，核心算法才是关键。论文告诉大家有 13 个小矮人，但是并没有告诉大家为什么是这 13 个小矮人。这就涉及到应用的核心算法。从计算的角度看，这 13 个小矮人足够了。但是从应用软件功能的角度，涉及的算法成百上千，十分繁琐。】

## 参考文献

- [ABAQUS 2006] ABAQUS finite element analysis home page. <http://www.hks.com>
- [Adiletta et al 2002] M. Adiletta, M. Rosenbluth, D. Bernstein, G. Wolrich, and H. Wilkinson, "The Next Generation of the Intel IXP Network Processors," Intel Technology Journal, vol. 6, no. 3, pp. 6–18, Aug. 15, 2002.
- [Allen et al 2006] E. Allen, V. Luchango, J.-W. Maessen, S. Ryu, G. Steele, and S. Tobin-Hochstadt, The Fortress Language Specification, 2006. Available at <http://research.sun.com/projects/plrg/>
- [Altschul et al 1990] S.F. Altschul, W. Gish, W. Miller, E.W. Myers, and D.J. Lipman, "Basic local alignment search tool," Journal Of Molecular Biology, vol. 215, no. 3, 1990, pp. 403–410.
- [Alverson et al 1990] R. Alverson, D. Callahan, D. Cummings, B. Koblenz, A. Porterfield, and B. Smith, "The Tera Computer System," in Proceedings of the 1990 ACM International Conference on Supercomputing (SC'90), pp. 1–6, Jun. 1990.
- [Alverson et al 1999] G.A. Alverson, C.D. Callahan, II, S.H. Kahan, B.D. Koblenz, A. Porterfield, B.J. Smith, "Synchronization Techniques in a Multithreaded Environment," US patent 6862635.
- [Arnold 2005] J. Arnold, "S5: the architecture and development flow of a software configurable processor," in Proceedings of the IEEE International Conference on Field-Programmable Technology, Dec. 2005, pp.121–128.
- [Arvind et al 2005] Arvind, K. Asanovic, D. Chiou, J.C. Hoe, C. Kozyrakis, S. Lu, M. Oskin, D. Patterson, J. Rabaey, and J. Wawrzynek, "RAMP: Research Accelerator for Multiple Processors - A Community Vision for a Shared Experimental Parallel HW/SW Platform," U.C. Berkeley technical report, UCB/CSD- 05-1412, 2005.
- [Aspuru-Guzik et al 2005] A. Aspuru-Guzik, R. Salomon-Ferrer, B. Austin, R. Perusquia-Flores, M.A. Griffin, R.A. Oliva, D. Skinner, D. Domin, and W.A. Lester, Jr., "Zori 1.0: A Parallel Quantum Monte Carlo Electronic Package," Journal of Computational Chemistry, vol. 26, no. 8, Jun. 2005, pp. 856–862.
- [Bader and Madduri 2006] D.A. Bader and K. Madduri, "Designing Multithreaded Algorithms for BreadthFirst Search and st-connectivity on the Cray MTA-2," in Proceedings of the 35th International Conference on Parallel Processing (ICPP), Aug. 2006, pp. 523–530.
- [Barnes and Hut 1986] J. Barnes and P. Hut, "A Hierarchical  $O(n \log n)$  force calculation algorithm," Nature, vol. 324, 1986.

- [Bell and Newell 1970] G. Bell and A. Newell, "The PMS and ISP descriptive systems for computer structures," in Proceedings of the Spring Joint Computer Conference, AFIPS Press, 1970, pp. 351–374.
- [Bernholdt et al 2002] D.E. Bernholdt, W.R. Elsasif, J.A. Kohl, and T.G.W. Epperly, "A Component Architecture for High-Performance Computing," in Proceedings of the Workshop on Performance Optimization via High-Level Languages and Libraries (POHLL-02), Jun. 2002.
- [Berry et al 2006] J.W. Berry, B.A. Hendrickson, S. Kahan, P. Konecny, "Graph Software Development and Performance on the MTA-2 and Eldorado," presented at the 48th Cray Users Group Meeting, Switzerland, May 2006.
- [Bilmes et al 1997] J. Bilmes, K. Asanovic, C.W. Chin, J. Demmel, "Optimizing matrix multiply using PHiPAC: a Portable, High-Performance, ANSI C coding methodology," in Proceedings of the 11th International Conference on Supercomputing, Vienna, Austria, Jul. 1997, pp. 340–347.
- [Blackford et al 1996] L.S. Blackford, J. Choi, A. Cleary, A. Petitet, R.C. Whaley, J. Demmel, I. Dhillon, K. Stanley, J. Dongarra, S. Hammarling, G. Henry, and D. Walker, "ScaLAPACK: a portable linear algebra library for distributed memory computers - design issues and performance," in Proceedings of the 1996 ACM/IEEE conference on Supercomputing, Nov. 1996.
- [Blackford et al 2002] L.S. Blackford, J. Demmel, J. Dongarra, I. Du, S. Hammarling, G. Henry, M. Heroux, L. Kaufman, A. Lumsdaine, A. Petitet, R. Pozo, K. Remington, R.C. Whaley, "An updated set of basic linear algebra subprograms (BLAS)," ACM Transactions on Mathematical Software (TOMS), vol. 28, no. 2, Jun. 2002, pp. 135–151.
- [Borkar 1999] S. Borkar, "Design challenges of technology scaling," IEEE Micro, vol. 19, no. 4, Jul.–Aug. 1999, pp. 23–29.
- [Borkar 2005] S. Borkar, "Designing Reliable Systems from Unreliable Components: The Challenges of Transistor Variability and Degradation," IEEE Micro, Nov.–Dec. 2005, pp. 10–16.
- [Brunel et al 2000] J.-Y. Brunel, K.A. Vissers, P. Lieverse, P. van der Wolf, W.M. Kruijtzter, W.J.M. Smiths, G. Essink, E.A. de Kock, "YAPI: Application Modeling for Signal Processing Systems," in Proceedings of the 37th Conference on Design Automation (DAC '00), 2000, pp. 402–405.
- [Callahan et al 2004] D. Callahan, B.L. Chamberlain, and H.P. Zima. "The Cascade High Productivity Language," in Proceedings of the 9th International Workshop on High-Level Parallel Programming Models and Supportive Environments (HIPS 2004), IEEE Computer Society, Apr. 2004, pp. 52–60.
- [Chandrakasan et al 1992] A.P. Chandrakasan, S. Sheng, and R.W. Brodersen, "Low-power CMOS digital design," IEEE Journal of Solid-State Circuits, vol. 27, no. 4, 1992, pp. 473–484.

[Charles et al 2005] P. Charles, C. Donawa, K. Ebcioglu, C. Grothoff, A. Kielstra, C. von Praun, V. Saraswat, and V. Sarkar, “X10: An Object-Oriented Approach to Non-Uniform Cluster Computing,” in Proceedings of the 20th Annual ACM SIGPLAN Conference on Object Oriented Programming, Systems, Languages, and Applications (OOPSLA ’05), Oct. 2005.

[Chen 2006] Y.K. Chen, Private Communication, Jun. 2006.

[Chinnery 2006] D. Chinnery, Low Power Design Automation, Ph.D. dissertation, Department of Electrical Engineering and Computer Sciences, University of California, Berkeley, Berkeley, CA, 2006.

[Chong and Catanzaro 2006] J. Chong and B. Catanzaro, Excel spreadsheet

[Chung et al 2006] E.S. Chung, J.C. Hoe, and B. Falsafi, “ProtoFlex: Co-Simulation for Component-wise FPGA Emulator Development,” in the 2nd Workshop on Architecture Research using FPGA Platforms (WARFP 2006), Feb. 2006.

[Colella 2004] P. Colella, “Defining Software Requirements for Scientific Computing,” presentation, 2004.

[Cooley and Tukey 1965] J. Cooley and J. Tukey, “An algorithm for the machine computation of the complex Fourier series,” *Mathematics of Computation*, vol. 19, 1965, pp. 297–301.

[Cristianini and Shawe-Taylor 2000] N. Cristianini and J. Shawe-Taylor, *An Introduction to Support Vector Machines*, Cambridge University Press, Cambridge, 2000.

[Dally and Towles 2001] W.J. Dally and B. Towles, “Route Packets, Not Wires: On-Chip Interconnection Networks,” in Proceedings of the 38th Conference on Design Automation (DAC ’01), 2001, pp. 684–689.

[Dean and Ghemawat 2004] J. Dean and S. Ghemawat, “MapReduce: Simplified Data Processing on Large Clusters,” in Proceedings of OSDI ’04: 6th Symposium on Operating System Design and Implementation, San Francisco, CA, Dec. 2004.

[Deitz 2005] S.J. Deitz, High-Level Programming Language Abstractions for Advanced and Dynamic Parallel Computations, PhD thesis, University of Washington, Feb. 2005.

[Demmel et al 1999] J. Demmel, S. Eisenstat, J. Gilbert, X. Li, and J. Liu, “A supernodal approach to sparse partial pivoting,” *SIAM Journal on Matrix Analysis and Applications*, vol. 20, no. 3, pp. 720–755.

[Demmel et al 2002] J. Demmel, D. Bailey, G. Henry, Y. Hida, J. Iskandar, X. Li, W. Kahan, S. Kang, A. Kapur, M. Martin, B. Thompson, T. Tung, and D. Yoo, “Design, Implementation and Testing of Extended and Mixed Precision BLAS,” *ACM Transactions on Mathematical Software*,

vol. 28, no. 2, Jun. 2002, pp.152–205.

[Dubey 2005] P. Dubey, “Recognition, Mining and Synthesis Moves Computers to the Era of Tera,” *Technology@Intel Magazine*, Feb. 2005.

[Duda and Hart 1973] R. Duda and P. Hart, *Pattern Classification and Scene Analysis*, New York: Wiley, 1973.

[Eatherton 2005] W. Eatherton, “The Push of Network Processing to the Top of the Pyramid,” keynote address at Symposium on Architectures for Networking and Communications Systems, Oct. 26–28, 2005. Slides available at:  
<http://www.cesr.ncsu.edu/ancs/slides/eathertonKeynote.pdf>

[Ebcioglu et al 2006] K. Ebcioglu, V. Sarkar, T. El-Ghazawi, J. Urbanic, “An Experiment in Measuring the Productivity of Three Parallel Programming Languages,” in *Proceedings of the Second Workshop on Productivity and Performance in High-End Computing (P-PHEC 2005)*, Feb. 2005.

[Edinburg 2006] University of Edinburg, “QCD-on-a-chip, (QCDOC),”  
<http://www.pparc.ac.uk/roadmap/rmProject.aspx?q=82>

[EEMBC 2006] Embedded Microprocessor Benchmark Consortium. <http://www.eembc.org>

[FLUENT 2006] FIDAP finite element for computational fluid dynamics analysis home page.  
<http://www.fluent.com/software/fidap/index.htm>

[Frigo and Johnson 1998] M. Frigo and S.G. Johnson, “FFTW: An adaptive software architecture for the FFT,” in *Proceedings of the 1998 IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP ’98)*, Seattle, WA, May 1998, vol. 3, pp. 1381–1384.

[Frigo and Johnson 2005] M. Frigo and S.G. Johnson, “The Design and Implementation of FFTW3,” *Proceedings of the IEEE*, vol. 93, no. 2, 2005, pp. 216–231.

[Gelsinger 2001] P.P. Gelsinger, “Microprocessors for the new millennium: Challenges, opportunities, and new frontiers,” in *Proceedings of the International Solid State Circuits Conference (ISSCC)*, 2001, pp. 22–25.

[Gonzalez and Horowitz 1996] R. Gonzalez and M. Horowitz, “Energy dissipation in general purpose microprocessors,” *IEEE Journal of Solid-State Circuits*, vol. 31, no. 9, 1996, pp. 1277–1284.

[Goodale et al 2003] T. Goodale, G. Allen, G. Lanfermann, J. Masso, T. Radke, E. Seidel, and J. Shalf, “The cactus framework and toolkit: Design and applications,” in *Vector and Parallel Processing (VECPAR’2002)*, 5th International Conference, Springer, 2003.

[Gordon et al 2002] M.I. Gordon, W. Thies, M. Karczmarek, J. Lin, A.S. Meli, A.A. Lamb, C. Leger, J. Wong, H. Hoffmann, D. Maze, and S. Amarasinghe, "A Stream Compiler for Communication-Exposed Architectures," MIT Technology Memo TM-627, Cambridge, MA, Mar. 2002.

[Granlund et al 2006] T. Granlund et al. GNU Superoptimizer FTP site.  
ftp: //prep.ai.mit.edu/pub/gnu/superopt

[Gries 2004] M. Gries, "Methods for Evaluating and Covering the Design Space during Early Design Development," *Integration, the VLSI Journal*, Elsevier, vol. 38, no. 2, Dec. 2004, pp. 131–183.

[Gries and Keutzer 2005] M. Gries and K. Keutzer (editors), *Building ASIPs: The MESCAL Methodology*, Springer, 2005.

[Gursoy and Kale 2004] A. Gursoy and L.V. Kale, "Performance and Modularity Benefits of MessageDriven Execution," *Journal of Parallel and Distributed Computing*, vol. 64, no. 4, Apr. 2004, pp. 461–480.

[Hammond et al 2004] L. Hammond, V. Wong, M. Chen, B. Hertzberg, B. Carlstrom, M. Prabhu, H. Wijaya, C. Kozyrakis, and K. Olukotun, "Transactional Memory Coherence and Consistency (TCC)," in *Proceedings of the 31st Annual International Symposium on Computer Architecture (ISCA '04)*, Jun. 2004.

[Harstein and Puzak 2003] A. Harstein and T. Puzak, "Optimum Power/Performance Pipeline Depth," in *Proceedings of the 36th IEEE/ACM International Symposium on Microarchitecture (MICRO-36)*, Dec. 2003, pp. 117–126.

[Hauser and Wawrzynek 1997] J.R. Hauser and J. Wawrzynek, "GARP: A MIPS processor with a reconfigurable coprocessor," in *Proceedings of the IEEE Symposium on FPGAs for Custom Computing Machines*, Apr. 1997, pp. 12–21.

[Hennessy and Patterson 2007] J. Hennessy and D. Patterson, *Computer Architecture: A Quantitative Approach*, 4th edition, Morgan Kauffman, San Francisco, 2007.

[Heo and Asanovic 2004] S. Heo and K. Asanovic, "Power-Optimal Pipelining in Deep Submicron Technology," in *Proceedings of the International Symposium on Low Power Electronics and Design*, 2004, pp. 218–223.

[Herlihy and Moss 1993] M. Herlihy and J.E.B. Moss, "Transactional Memory: Architectural Support for Lock-Free Data Structures," in *Proceedings of the 20th Annual International Symposium on Computer Architecture (ISCA '93)*, 1993, pp. 289–300.



[Hewitt et al 1973] C. Hewitt, P. Bishop, and R. Steiger, "A Universal Modular Actor Formalism for Artificial Intelligence," in Proceedings of the 1973 International Joint Conference on Artificial Intelligence, 1973, pp. 235–246.

[Hillis and Tucker 1993] W.D. Hillis and L.W. Tucker, "The CM-5 Connection Machine: A Scalable Supercomputer," Communications of the ACM, vol. 36, no. 11, Nov. 1993, pp. 31–40.

[Hochstein et al 2005] L. Hochstein, J. Carver, F. Shull, S. Asgari, V.R. Basili, J.K. Hollingsworth, M. Zelkowitz. "Parallel Programmer Productivity: A Case Study of Novice Parallel Programmers," International Conference for High Performance Computing, Networking and Storage (SC'05). Nov. 2005.

[Horowitz 2006] M. Horowitz, personal communication and Excel spreadsheet.

[Hrishikesh et al 2002] M.S. Hrishikesh, D. Burger, N.P. Jouppi, S.W. Keckler, K.I. Farkas, and P. Shivakumar, "The Optimal Logic Depth Per Pipeline Stage is 6 to 8 FO4 Inverter Delays," in Proceedings of the 29th Annual International Symposium on Computer Architecture (ISCA '02), May 2002, pp. 14–24.

[HPCS 2006] DARPA High Productivity Computer Systems home page. <http://www.highproductivity.org/>

[IBM 2006] IBM Research, "MD-GRAPE." <http://www.research.ibm.com/grape/>

[Im et al 2005] E.J. Im, K. Yelick, and R. Vuduc, "Sparsity: Optimization framework for sparse matrix kernels," International Journal of High Performance Computing Applications, vol. 18, no. 1, Spr. 2004, pp. 135–158.

[Intel 2004] Intel Corporation, "Introduction to Auto-Partitioning Programming Model," Literature number 254114-001, 2004.

[Kamil et al 2005] S.A. Kamil, J. Shalf, L. Oliker, and D. Skinner, "Understanding Ultra-Scale Application Communication Requirements," in Proceedings of the 2005 IEEE International Symposium on Workload Characterization (IISWC), Austin, TX, Oct. 6–8, 2005, pp. 178–187. (LBNL-58059)

[Kantowitz and Sorkin 1983] B.H. Kantowitz and R.D. Sorkin, Human Factors: Understanding People/System Relationships, New York, NY, John Wiley & Sons, 1983.

[Killian et al 2001] E. Killian, C. Rowen, D. Maydan, and A. Wang, "Hardware/Software Instruction set Configurability for System-on-Chip Processors," in Proceedings of the 38th Conference on Design Automation (DAC '01), 2001, pp. 184–188.

[Koelbel et al 1993] C.H. Koelbel, D.B. Loveman, R.S. Schreiber, G.L. Steele Jr., and M.E. Zosel,

The High Performance Fortran Handbook, The MIT Press, 1993. ISBN 0262610949.

[Kozyrakis and Olukotun 2005] C. Kozyrakis and K. Olukotun, "ATLAS: A Scalable Emulator for Transactional Parallel Systems," in Workshop on Architecture Research using FPGA Platforms, 11th International Symposium on High-Performance Computer Architecture (HPCA-11 2005), San Francisco, CA, Feb. 13, 2005.

[Kumar et al 2003] R. Kumar, K.I. Farkas, N.P. Jouppi, P. Ranganathan, and D.M. Tullsen, "Single-ISA Heterogeneous Multi-core Architectures: The Potential for Processor Power Reduction," in Proceedings of the 36th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO-36), Dec. 2003.

[Kuo et al 2005] K. Kuo, R.M. Rabbah, and S. Amarasinghe, "A Productive Programming Environment for Stream Computing," in Proceedings of the Second Workshop on Productivity and Performance in HighEnd Computing (P-PHEC 2005), Feb. 2005.

[Kuon and Rose 2006] I. Kuon and J. Rose, "Measuring the gap between FPGAs and ASICs," in Proceedings of the International Symposium on Field Programmable Gate Arrays (FPGA '06), Monterey, California, USA, ACM Press, New York, NY, Feb. 22–24, 2006, pp. 21–30.

[Massalin 1987] H. Massalin, "Superoptimizer: a look at the smallest program," in Proceedings of the Second International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS II), Palo Alto, CA, 1987, pp. 122–126.

[MathWorks 2004] The MathWorks, "Real-Time Workshop 6.1 Datasheet," 2004.

[MathWorks 2006] The MathWorks, MATLAB Function Reference, 2006.

[Mattson 1999] T. Mattson, "A Cognitive Model for Programming," U. Florida whitepaper, 1999. Available at <http://www.cise.ufl.edu/research/ParallelPatterns/PatternLanguage/Background/Psychology/CognitiveModel.html>

[Monaghan 1982] J.J. Monaghan, "Shock Simulation by the Particle Method SPH," Journal of Computational Physics, vol. 52, 1982, pp. 374–389.

[Mukherjee et al 2005] S.S. Mukherjee, J. Emer, and S.K. Reinhardt, "The Soft Error Problem: An Architectural Perspective," in Proceedings of the 11th International Symposium on High-Performance Computer Architecture (HPCA-11 2005), Feb. 2005, pp. 243–247.

[Nyberg et al 2004] C. Nyberg, J. Gray, C. Koester, "A Minute with Nsort on a 32P NEC Windows Itanium2 Server", <http://www.ordinal.com/NsortMinute.pdf>, 2004.

[Opencores 2006] Opencores home page. <http://www.opencores.org>

[OpenMP 2006] OpenMP home page. <http://www.openmp.org>

[OpenSPARC 2006] OpenSPARC home page. <http://opensparc.sunsource.net>

[OSKI 2006] OSKI home page. <http://bebop.cs.berkeley.edu/oski/about.html>

[Pancake and Bergmark 1990] C.M. Pancake and D. Bergmark, "Do Parallel Languages Respond to the Needs of Scientific Programmers?" IEEE Computer, vol. 23, no. 12, Dec. 1990, pp. 13–23.

[Patterson 2004] D. Patterson, "Latency Lags Bandwidth," Communications of the ACM, vol. 47, no. 10, Oct. 2004, pp. 71–75.

[Patterson et al 1997] D. Patterson, T. Anderson, N. Cardwell, R. Fromm, K. Keeton, C. Kozyrakis, R. Thomas, and K. Yelick, "A Case for Intelligent RAM: IRAM," IEEE Micro, vol. 17, no. 2, Mar.–Apr. 1993, pp. 34–44.

[Paulin 2006] P. Paulin, personal communication and Excel spreadsheet.

[Plishker et al 2004] W. Plishker, K. Ravindran, N. Shah, K. Keutzer, "Automated Task Allocation for Network Processors," in Network System Design Conference Proceedings, Oct. 2004, pp. 235–245.

[Poole et al 1998] D. Poole, A. Mackworth and R. Goebel, Computational Intelligence: A Logical Approach, Oxford University Press, New York, 1998.

[Power.org 2006] Power.org home page. <http://www.power.org>

[Pthreads 2004] IEEE Std 1003.1-2004, The Open Group Base Specifications Issue 6, section 2.9, IEEE and The Open Group, 2004.

[Pyla et al 2004] P.S. Pyla, M.A. Perez-Quinones, J.D. Arthur, H.R. Hartson, "What we should teach, but don't: Proposal for cross pollinated HCI-SE Curriculum," in Proceedings of ASEE/IEEE Frontiers in Education Conference, Oct. 2004, pp. S1H/17–S1H/22.

[Rajwar and Goodman 2002] R. Rajwar and J. R. Goodman, "Transactional lock-free execution of lockbased programs," in Proceedings of the 10th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS X), ACM Press, New York, NY, USA, Oct. 2002, pp. 5–17.

[Reason 1990] J. Reason, Human error, New York, Cambridge University Press, 1990.

[Rosenblum 2006] M. Rosenblum, “The Impact of Virtualization on Computer Architecture and Operating Systems,” Keynote Address, 12th International Conference on Architectural Support for Programming

Languages and Operating Systems (ASPLOS XII), San Jose, California, Oct. 23, 2006.

[Rowen and Leibson 2005] C. Rowen and S. Leibson, Engineering the Complex SOC : Fast, Flexible Design with Configurable Processors, Prentice Hall, 2nd edition, 2005.

[Scott 1996] S.L. Scott. “Synchronization and communication in the T3E multiprocessor.” In Proceeding of the Seventh International Conference on Architectural Support for Programming Languages and

Operating Systems (ASPLOS VII), Cambridge, MA, Oct. 1996.

[Seffah 2003] A. Seffah, “Learning the Ropes: Human-Centered Design Skills and Patterns for Software

Engineers’ Education,” Interactions, vol. 10, 2003, pp. 36–45.

[Shah et al 2004a] N. Shah, W. Plishker, K. Ravindran, and K. Keutzer, “NP-Click: A Productive Software Development Approach for Network Processors,” IEEE Micro, vol. 24, no. 5, Sep. 2004, pp. 45–54.

[Shah et al 2004b] N. Shah, W. Plishker, and K. Keutzer, “Comparing Network Processor Programming Environments: A Case Study,” 2004 Workshop on Productivity and Performance in High-End Computing (P-PHEC), Feb. 2004.

[Shalf et al 2005] J. Shalf, S.A. Kamil, L. Oliker, and D. Skinner, “Analyzing Ultra-Scale Application Communication Requirements for a Reconfigurable Hybrid Interconnect,” in Proceedings of the 2005 ACM/IEEE Conference on Supercomputing (SC ’05), Seattle, WA, Nov. 12–18, 2005. (LBNL-58052)

[Singh et al 1992] J.P. Singh, W.-D. Weber, and A. Gupta, “SPLASH: Stanford Parallel Applications for Shared-Memory,” in Computer Architecture News, Mar. 1992, vol. 20, no. 1, pages 5–44.

[Snir et al 1998] M. Snir, S. Otto, S. Huss-Lederman, D. Walker, and J. Dongarra. MPI: The Complete Reference (Vol. 1). The MIT Press, 1998. ISBN 0262692155.

[Solar-Lezama et al 2005] A. Solar-Lezama, R. Rabbah, R. Bodik, and K. Ebcioglu, “Programming by Sketching for Bit-Streaming Programs,” in Proceedings of the 2005 ACM SIGPLAN conference on Programming language design and implementation (PLDI’05), Jun. 2005, pp. 281–294.

[Soteriou et al 2006] V. Soteriou, H. Wang, L.-S. Peh, “A Statistical Traffic Model for On-Chip Interconnection Networks,” in Proceedings of the 14th IEEE International Symposium on

Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS '06), Sep. 2006, pp. 104–116.

[SPEC 2006] Standard Performance Evaluation Corporation (SPEC). <http://www.spec.org/index.html>

[Srinivasan et al 2002] V. Srinivasan, D. Brooks, M. Gschwind, P. Bose, V. Zyuban, P.N. Strenski, and P.G. Emma, “Optimizing pipelines for power and performance,” in Proceedings of the 35th International Symposium on Microarchitecture (MICRO-35), 2002, pp. 333–344.

[Sterling 2006] T. Sterling, “Multi-Core for HPC: Breakthrough or Breakdown?” Panel discussion at the International Conference for High Performance Computing, Networking and Storage (SC'06), Nov.2006..Slides available at <http://www.cct.lsu.edu/~tron/SC06.html>

[Sylvester et al 1999] D. Sylvester, W. Jiang, and K. Keutzer, “Berkeley Advanced Chip Performance Calculator,” <http://www.eecs.umich.edu/~dennis/bacpac/index.html>

[Sylvester and Keutzer 1998] D. Sylvester and K. Keutzer, “Getting to the Bottom of Deep Submicron,” in Proceedings of the International Conference on Computer-Aided Design, Nov. 1998, pp. 203–211.

[Sylvester and Keutzer 2001] D. Sylvester and K. Keutzer, “Microarchitectures for systems on a chip in small process geometries,” Proceedings of the IEEE, Apr. 2001, pp. 467–489.

[Teja 2003] Teja Technologies, “Teja NP Datasheet,” 2003.

[Teragrid 2006] NSF Teragrid home page. <http://www.teragrid.org/>

[Tokyo 2006] University of Tokyo, “GRAPE,” <http://grape.astron.s.u-tokyo.ac.jp>

[Vadhiyar et al 2000] S. Vadhiyar, G. Fagg, and J. Dongarra, “Automatically Tuned Collective Communications,” in Proceedings of the 2000 ACM/IEEE Conference on Supercomputing, Nov. 2000.

[Vahala et al 2005] G. Vahala, J. Yepez, L. Vahala, M. Soe, and J. Carter, “3D entropic lattice Boltzmann simulations of 3D Navier-Stokes turbulence,” in Proceedings of the 47th Annual Meeting of the APS Division of Plasma Physics, 2005.

[Vetter and McCracken 2001] J.S. Vetter and M.O. McCracken, “Statistical Scalability Analysis of Communication Operations in Distributed Applications,” in Proceedings of the Eighth ACM SIGPLAN Symposium on Principles and Practices of Parallel Programming (PPOPP), 2001, pp.

123–132.

[Vetter and Mueller 2002] J.S. Vetter and F. Mueller, “Communication Characteristics of Large-Scale Scientific Applications for Contemporary Cluster Architectures,” in Proceedings of the 16th International Parallel and Distributed Processing Symposium (IPDPS), 2002, pp. 272–281.

[Vetter and Yoo 2002] J.S. Vetter and A. Yoo, “An Empirical Performance Evaluation of Scalable Scientific Applications,” in Proceedings of the 2002 ACM/IEEE Conference on Supercomputing, 2002.

[Vuduc et al 2006] R. Vuduc, J. Demmel, and K. Yelick, “OSKI: Optimized Sparse Kernel Interface,”  
[http: //bebop.cs.berkeley.edu/oski/](http://bebop.cs.berkeley.edu/oski/).

[Warren 2006] H. Warren, A Hacker’s Assistant. [http: //www.hackersdelight.org](http://www.hackersdelight.org)

[Wawrzynek et al 2006] J. Wawrzynek, D. Patterson, M. Oskin, S.-L. Lu, C. Kozyrakis, J.C. Joe, D. Chiou, and K. Asanovic, “RAMP: A Research Accelerator for Multiple Processors,” U.C. Berkeley technical report, 2006.

[Weinburg 2004] B. Weinberg, “Linux is on the NPU control plane,” EE Times, Feb. 9, 2004.

[Whaley and Dongarra 1998] R.C. Whaley and J.J. Dongarra, “Automatically tuned linear algebra software,” in Proceedings of the 1998 ACM/IEEE Conference on Supercomputing, 1998.

[Van der Wijngaart 2002] R.F. Van der Wijngaart, “NAS Parallel Benchmarks Version 2.4,” NAS technical report, NAS-02-007, Oct. 2002.

[Wind River 2006] Wind River home page. [http: //www.windriver.com/products/platforms/general\\_purpose/index.html](http://www.windriver.com/products/platforms/general_purpose/index.html)

[Wolfe 2004] A. Wolfe, “Intel Clears Up Post-Tejas Confusion,” VARBusiness, May 17, 2004.  
[http: //www.varbusiness.com/sections/news/breakingnews.jhtml?articleId=18842588](http://www.varbusiness.com/sections/news/breakingnews.jhtml?articleId=18842588)

[Woo et al 1995] S.C. Woo, M. Ohara, E. Torrie, J.P. Singh, and A. Gupta, “The SPLASH-2 Programs: Characterization and Methodological Considerations,” in Proceedings of the 22nd International Symposium on Computer Architecture (ISCA ’95), Santa Margherita Ligure, Italy, Jun. 1995, pp. 24–36.

[Wulf and McKee 1995] W.A. Wulf and S.A. McKee, “Hitting the Memory Wall: Implications of the Obvious,” Computer Architecture News, vol. 23, no. 1, Mar. 1995, pp. 20–24.

[Xu et al 2003] M. Xu, R. Bodik, and M.D. Hill, “A ‘Flight Data Recorder’ for Enabling

Full-System Multiprocessor Deterministic Replay,” in Proceedings of the 31st Annual International Symposium on Computer Architecture (ISCA '04), 2004.

[Zarlink 2006] Zarlink, “PDSP16515A Stand Alone FFT Processor,” [http://products.zarlink.com/product\\_profiles/PDSP16515A.htm](http://products.zarlink.com/product_profiles/PDSP16515A.htm).