Roofline: An Insightful Visual Performance Model for Floating-Point Programs and Multicore Architectures*

Samuel Williams, Andrew Waterman, and David Patterson

Parallel Computing Laboratory, 565 Soda Hall, U.C. Berkeley, Berkeley, CA 94720-1776, 510-642-6587
samw, waterman, pattrsn@eecs.berkeley.edu

ABSTRACT

We propose an easy-to-understand, visual performance model that offers insights to programmers and architects on improving parallel software and hardware for floating point computations.

1. INTRODUCTION

Conventional wisdom in computer architecture led to homogeneous designs. Nearly every desktop and server computer uses caches, pipelining, superscalar instruction issue, and out-of-order execution. Although the instruction sets varied, the microprocessors were all from the same school of design.

The switch to multicore means that microprocessors will become more diverse, since there is no conventional wisdom yet for them. For example, some offer many simple processors versus fewer complex processors, some depend on multithreading, and some even replace caches with explicitly addressed local stores. Manufacturers will likely offer multiple products with differing number of cores to cover multiple price-performance points, since the cores per chip will likely double every two years [4].

While diversity may be understandable in this time of uncertainty, it exacerbates the already difficult job of programmers, compiler writers, and even architects. Hence, an easy-to-understand model that offers performance guidelines could be especially valuable.

A model need not be perfect, just insightful. For example, the 3Cs model for caches is an analogy [19]. It is not a perfect model, since it ignores potentially important factors like block size, block allocation policy, and block replacement policy. Moreover, it has quirks. For example, a miss can be labeled capacity in one design and conflict in another cache of the same size. Yet, the 3Cs model has been popular for nearly 20 years because it offers insights into the behavior of programs, helping programmers, compiler writers, and architects improve their respective designs.

This paper proposes such a model and demonstrates it on four diverse multicore computers using four key floating-point kernels.

2. PERFORMANCE MODELS

Stochastic analytical models [14][28] and statistical performance models [7][27] can predict program performance on multiprocessors accurately. However, they rarely provide insights into how to improve performance of programs, compilers, or computers [1] or they can be hard to use by non-experts [27].

An alternative, simpler approach is *bound and bottleneck* analysis. Instead of trying to predict performance, it provides [20]

"valuable insight into the primary factors affecting the performance of computer systems. In particular, the critical influence of the system bottleneck is highlighted and quantified."

The best-known example is surely Amdahl's Law [3], which states simply that the performance gain of a parallel computer is limited by the serial portion of a parallel program. It has been recently applied to heterogeneous multicore computers [4][18].

3. THE ROOFLINE MODEL

We believe that for the recent past and foreseeable future, off-chip memory bandwidth will often be the constraining resource[23]. Hence, we want a model that relates processor performance to off-chip memory traffic.

Towards that goal, we use the term *operational intensity* to mean operations per byte of DRAM traffic. We define total bytes accessed as those that go to the main memory *after* they have been filtered by the cache hierarchy. That is, we measure traffic between the caches and memory rather than between the processor and the caches. Thus, *operational intensity* suggests the DRAM bandwidth needed by a kernel on a particular computer.

We use operational intensity instead of the terms *arithmetic intensity* [16] or *machine balance* [8][11] for two reasons. First, arithmetic intensity and machine balance measure traffic between the processor and cache, whereas we want to measure traffic between the caches and DRAM. This subtle change allows us to include memory optimizations of a computer into our bound and bottleneck model. Second, we think the model will work with kernels where the operations are not arithmetic (see Section 7), so we needed a more general term than arithmetic.

The proposed model ties together floating-point performance, operational intensity, and memory performance together in a two-dimensional graph. Peak floating-point performance can be found using the hardware specifications or microbenchmarks. The working sets of the kernels we consider here do not fit fully in on-chip caches, so peak memory performance is defined by the memory system behind the caches. Although you can find memory performance with the STREAM benchmark [22], for this work we wrote a series of progressively optimized microbenchmarks designed to determine sustainable DRAM bandwidth. They include all techniques to get the best memory performance, including prefetching and data alignment. (Section A.1 in the Appendix gives a more details of how to measure processor and memory performance and operational intensity.)[1]

Figure 1a shows the model for a 2.2 GHz AMD Opteron X2 model 2214 in a dual socket system. The graph is on a log-log scale. The Y-axis is attainable floating-point performance. The X-axis is operational intensity, varying from 1/4 Flops/DRAM byte accessed to 16 Flops/DRAM byte accessed. The system being

[1] Appendix A of this paper is on the CACM web site.

*To appear in *Communications of the ACM* April 2008. Revised August and October 2008.

屋顶线：浮点程序和多核架构的深刻视觉性能模型*

塞缪尔·威廉姆斯，安德鲁·沃特曼，和大卫·帕特森
平行计算实验室，565 Soda Hall，伯克利加州大学，伯克利，加州 94720-1776，510-642-6587
samw, waterman, pattrsn@eecs.berkeley.edu

摘要
我们提出了一种易于理解的可视化性能模型，为程序员和架构师提供了关于改进浮点计算的并行软件和硬件的见解。

1. 引言
传统的计算机架构理念导致了同质化设计。几乎每台桌面和服务器计算机都使用缓存、流水线、超标量指令发射和乱序执行。尽管指令集有所不同，但微处理器都来自同一设计流派。

切换到多核意味着微处理器将变得更加多样化，因为目前还没有关于它们的传统智慧。例如，有些提供许多简单的处理器，而有些则提供较少的复杂处理器，有些依赖于多线程，还有一些甚至用显式寻址的本地存储替代缓存。

制造商可能会提供多种产品，具有不同数量的核心，以覆盖多个性价比点，因为每个芯片的核心数量可能每两年翻一番 [4]。

虽然在这个不确定的时期多样性是可以理解的，但它加剧了程序员、编译器编写者甚至架构师已经艰难的工作。因此，一个易于理解的模型，提供性能指导，可能特别有价值。

一个模型不必完美，只需具有洞察力。例如，缓存的3Cs模型是一个类比[19]。它不是一个完美的模型，因为它忽略了诸如块大小、块分配策略和块替换策略等潜在重要因素。此外，它还有一些怪癖。例如，在一个设计中，未命中可以被标记为容量，而在另一个相同大小的缓存中则可以标记为冲突。然而，3Cs模型在近20年内一直很受欢迎，因为它提供了对程序行为的洞察，帮助程序员、编译器作者和架构师改进各自的设计。

本文提出了这样一个模型，并在四个不同的多核计算机上使用四个关键的浮点内核进行了演示。

2. 性能模型
随机分析模型 [14][28] 和统计性能模型 [7][27] 可以准确预测多处理器上的程序性能。然而，它们很少提供关于如何提高程序、编译器或计算机性能的见解 [1]，或者对于非专家来说可能难以使用 [27]。

一种替代的、更简单的方法是 *bound and bottleneck* 分析。它不是试图预测性能，而是提供 [20]

“valuable insight into the primary factors affecting the performance of computer systems. In particular, the critical influence of the system bottleneck is highlighted and quantified.”

最著名的例子无疑是阿姆达尔定律 [3]，它简单地表明并行计算机的性能提升受到并行程序中串行部分的限制。它最近已被应用于异构多核计算机 [4][18]。

3. 屋顶线模型
我们相信，在最近的过去和可预见的未来，芯片外存储带宽通常将是限制资源[23]。因此，我们希望有一个模型将处理器性能与芯片外存储流量相关联。

为了实现这个目标，我们使用术语 *operational intensity* 来表示每字节 DRAM 流量的操作数。我们将总访问字节定义为那些经过缓存层次过滤后进入主内存的字节 *after*。也就是说，我们测量的是缓存与内存之间的流量，而不是处理器与缓存之间的流量。因此，*operational intensity* 表示特定计算机上内核所需的 DRAM 带宽。

我们使用操作强度而不是术语 *arithmetic intensity* [16] 或 *machine balance* [8][11]，原因有两个。首先，算术强度和机器平衡测量处理器与缓存之间的流量，而我们想要测量缓存与 DRAM 之间的流量。这一微妙的变化使我们能够将计算机的内存优化纳入我们的界限和瓶颈模型。其次，我们认为该模型将适用于操作不是算术的内核（见第 7 节），因此我们需要一个比算术更通用的术语。

所提出的模型将浮点性能、操作强度和内存性能结合在一个二维图中。可以通过硬件规格或微基准测试找到峰值浮点性能。我们在这里考虑的内核的工作集并不能完全适应片上缓存，因此峰值内存性能由缓存后面的内存系统定义。尽管可以通过STREAM基准测试[22]找到内存性能，但在这项工作中，我们编写了一系列逐步优化的微基准测试，旨在确定可持续的DRAM带宽。它们包括获取最佳内存性能的所有技术，包括预取和数据对齐。（附录A.1节提供了有关如何测量处理器和内存性能及操作强度的更多细节。）[1]

图 1a 显示了在双插槽系统中 2.2 GHz AMD Opteron X2 2214 型号的模型。该图是对数-对数尺度。Y 轴是可达到的浮点性能。X 轴是操作强度，从 1/4 Flops/DRAM 字节访问到 16 Flops/DRAM 字节访问。该系统正在

[1] Appendix A of this paper is on the CACM web site.

*To appear in *Communications of the ACM* April 2008. Revised August and October 2008.

modeled has a peak double precision floating-point performance of 17.6 GFlops/sec and a peak memory bandwidth of 15 GBytes/sec from our benchmark. This latter measure is the steady state bandwidth potential of the memory in a computer, not the pin bandwidth of the DRAM chips.

We can plot a horizontal line showing peak floating-point performance of the computer. Obviously, the actual floating-point performance of a floating-point kernel can be no higher than the horizontal line, since that is a hardware limit.

How could we plot the peak memory performance? Since X-axis is GFlops per byte and the Y-axis is GFlops per second, bytes per second—which equals (GFlops/second)/(GFlops/byte)—is just a line at a 45-degree angle in this figure. Hence, we can plot a second line that gives the maximum floating-point performance that the memory system of that computer can support for a given operational intensity. This formula drives the two performance limits in the graph in Figure 1a:

Attainable GFlops/sec = Min(Peak Floating Point Performance,
Peak Memory Bandwidth x Operational Intensity)

These two lines intersect at the point of peak computational performance and peak memory bandwidth. Note that these limits are created once per multicore computer, not once per kernel.

For a given kernel, we can find a point on the X-axis based on its operational intensity. If we draw a (pink dashed) vertical line through that point, the performance of the kernel on that computer must lie somewhere along that line.

The horizontal and diagonal lines give this bound model its name. The *Roofline* sets an upper bound on performance of a kernel depending on its operational intensity. If we think of operational intensity as a column that hits the roof, either it hits the flat part of the roof, which means performance is compute bound, or it hits the slanted part of the roof, which means performance is ultimately memory bound. In Figure 1a, a kernel with operational intensity 2 is compute bound and a kernel with operational intensity 1 is memory bound. Given a Roofline, you can use it repeatedly on different kernels, since the Roofline doesn't vary.

Note that the *ridge point*, where the diagonal and horizontal roofs meet, offers an insight into the overall performance of the computer. The x-coordinate of the ridge point is the minimum operational intensity required to achieve maximum performance. If the ridge point is far to the right, then only kernels with very high operational intensity can achieve the maximum performance of that computer. If it is far to the left, then almost any kernel can potentially hit the maximum performance. As we shall see (Section 6.3.5), the ridge point suggests the level of difficulty for programmers and compiler writers to achieve peak performance.

To illustrate, let's compare the Opteron X2 with two cores in Figure 1a to its successor, the Opteron X4 with four cores. To simplify board design, they share the same socket. Hence, they have the same DRAM channels and can thus have the same peak memory bandwidth, although the prefetching is better in the X4. In addition to doubling the number of cores, the X4 also has twice the peak floating-point performance per core: X4 cores can issue two floating-point SSE2 instructions per clock cycle while X2 cores can issue two every other clock. As the clock rate is slightly faster—2.2 GHz for X2 versus 2.3 GHz for X4—the X4 has slightly more than four times the peak floating-point performance of the X2 with the same memory bandwidth.

Figure 1b compares the Roofline models for both systems. As expected, the ridge point shifts right from 1.0 in the Opteron X2 to 4.4 in the Opteron X4. Hence, to see a performance gain in the X4, kernels need an operational intensity higher than 1.

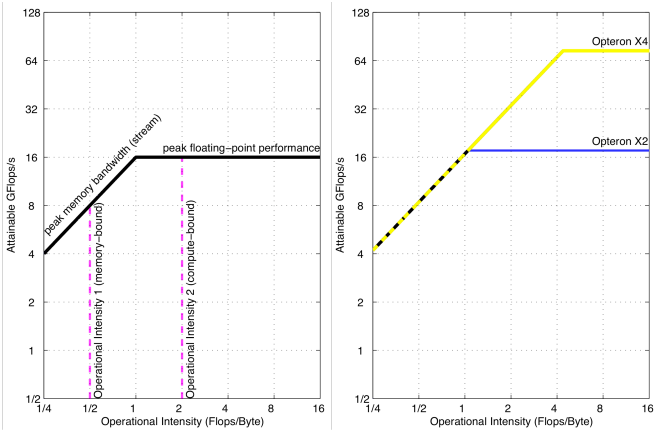

Figure 1. Roofline Model for (a) AMD Opteron X2 on left and (b) Opteron X2 vs. Opteron X4 on right.

4. ADDING CEILINGS TO THE MODEL

The Roofline model gives an upper bound to performance. Suppose your program is performing far below its Roofline. What optimizations should you perform, and in what order? Another advantage of bound and bottleneck analysis is [20]

"a number of alternatives can be treated together, with a single bounding analysis providing useful information about them all."

We leverage this insight to add multiple *ceilings* to the Roofline model to guide which optimizations to perform, which are similar to the guidelines that loop balance gives the compiler. We can think of each of these optimizations as a "performance ceiling" below the appropriate Roofline, meaning that you cannot break through a ceiling without performing the associated optimization.

For example, to reduce computational bottlenecks on the Opteron X2, two optimizations can help almost any kernel:

1. *Improve instruction level parallelism (ILP) and apply SIMD*. For superscalar architectures, the highest performance comes when fetching, executing, and committing the maximum number of instructions per clock cycle. The goal here is to improve the code from the compiler to increase ILP. The highest performance comes from completely covering the functional unit latency. One way is by unrolling loops. For the x86-based architectures, another way is using floating-point SIMD instructions whenever possible, since an SIMD instruction operates on pairs of adjacent operands.

2. *Balance floating-point operation mix*. The best performance requires that a significant fraction of the instruction mix be floating-point operations (see Section 7). Peak floating-point

建模的峰值双精度浮点性能为 17.6 GFlops/sec，峰值内存带宽为 15 GBytes/sec，来自我们的基准测试。后者是计算机内存的稳态带宽潜力，而不是 DRAM 芯片的引脚带宽。

我们可以绘制一条水平线，显示计算机的峰值浮点性能。显然，浮点内核的实际浮点性能不能高于水平线，因为那是硬件限制。

我们如何绘制峰值内存性能？由于X轴是每字节的GFlops，Y轴是每秒的GFlops，每秒的字节数——这等于(GFlops/秒)/(GFlops/字节)——在这个图中只是一个45度角的直线。因此，我们可以绘制第二条线，给出该计算机的内存系统在给定操作强度下可以支持的最大浮点性能。这个公式驱动了图1a中图表中的两个性能限制：

可达 GFlops/秒 = 最小（峰值浮点性能，峰值内存带宽 x 操作强度）

这两条线在峰值计算性能和峰值内存带宽的点相交。请注意，这些限制是在每个多核计算机上创建一次，而不是在每个内核上创建一次。

对于给定的内核，我们可以根据其操作强度在X轴上找到一个点。如果我们通过该点绘制一条（粉色虚线）垂直线，则该计算机上内核的性能必须位于该线的某个位置。

水平和对角线给这个边界模型命名。*Roofline* 设置了一个性能的上限，取决于其操作强度。如果我们将操作强度视为一个撞击屋顶的柱子，它要么撞击屋顶的平坦部分，这意味着性能受计算限制，要么撞击屋顶的倾斜部分，这意味着性能最终受内存限制。在图1a中，操作强度为2的内核是计算限制的，而操作强度为1的内核是内存限制的。给定一个屋顶线，你可以在不同的内核上反复使用它，因为屋顶线是恒定的。

请注意，*ridge point* 的位置，即对角线和水平屋顶相交的地方，提供了对计算机整体性能的洞察。山脊点的 x 坐标是实现最大性能所需的最小操作强度。如果山脊点位于右侧，则只有具有非常高操作强度的内核才能实现该计算机的最大性能。如果它位于左侧，则几乎任何内核都有可能达到最大性能。正如我们将在第 6.3.5 节中看到的，山脊点暗示了程序员和编译器作者实现峰值性能的难度级别。

为了说明这一点，让我们将图1a中的两个核心的Opteron X2与其后继者四核心的Opteron X4进行比较。为了简化电路板设计，它们共享相同的插槽。因此，它们具有相同的DRAM通道，因此可以具有相同的峰值内存带宽，尽管X4的预取性能更好。除了将核心数量翻倍外，X4每个核心的峰值浮点性能也翻倍：X4核心每个时钟周期可以发出两个浮点SSE2指令，而X2则不行。

核心可以每隔一个时钟发出两个指令。由于时钟频率略快——X2为2.2 GHz，而X4为2.3 GHz——X4的峰值浮点性能略高于X2的四倍，且内存带宽相同。

图 1b 比较了两个系统的 Roofline 模型。正如预期的那样，山脊点从 Opteron X2 的 1.0 向右移动到 Opteron X4 的 4.4。因此，要在 X4 中看到性能提升，内核需要的操作强度高于 1。

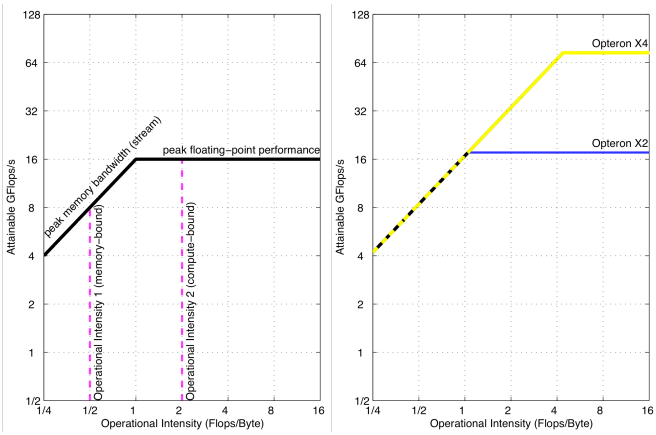

图 1. 左侧为 (a) AMD Opteron X2 的屋顶线模型和 (b) Opteron X2 与右侧的 Opteron X4。

4. 将天花板添加到模型中

屋顶线模型为性能提供了一个上限。假设您的程序的性能远低于其屋顶线。您应该进行哪些优化，以及以什么顺序进行？界限和瓶颈分析的另一个优点是 [20]

“a number of alternatives can be treated together, with a single bounding analysis providing useful information about them all.”

我们利用这一见解在 Roofline 模型中添加多个 *ceilings*，以指导应执行哪些优化，这与循环平衡给编译器的指导原则类似。我们可以将这些优化视为适当 Roofline 之下的“性能上限”，这意味着在未执行相关优化的情况下，无法突破这一上限。

例如，为了减少 Opteron X2 上的计算瓶颈，两个优化几乎可以帮助任何内核：

1. *Improve instruction level parallelism (ILP) and apply SIMD.* 对于超标量架构，最高性能来自于在每个时钟周期内获取、执行和提交最大数量的指令。这里的目标是改善编译器生成的代码，以增加指令级并行性（ILP）。最高性能来自于完全覆盖功能单元的延迟。实现这一目标的一种方法是展开循环。对于基于x86的架构，另一种方法是尽可能使用浮点SIMD指令，因为SIMD指令对相邻操作数对进行操作。

2. *Balance floating-point operation mix.* 最佳性能要求指令组合中有相当一部分为浮点运算（见第7节）。峰值浮点

performance typically also requires an equal number of simultaneous floating-point additions and multiplications, since many computers have multiply-add instructions or because they have an equal number of adders and multipliers.

To reduce memory bottlenecks, three optimizations can help:

3. *Restructure loops for unit stride accesses*. Optimizing for unit stride memory accesses engages hardware prefetching, which significantly increases memory bandwidth.

4. *Ensure memory affinity*. Most microprocessors today include a memory controller on the same chip with the processors. If the system has two multicore chips, then some addresses go to the DRAM local to one multicore chip and the rest must go over a chip interconnect to access the DRAM that is local to another chip. This latter case lowers performance. This optimization allocates data and the threads tasked to that data to the same memory-processor pair, so that the processors rarely have to access the memory attached to other chips.

5. *Use software prefetching*. Usually the highest performance requires keeping many memory operations in flight, which is easier to do via prefetching rather than waiting until the data is actually requested by the program. On some computers, software prefetching delivers more bandwidth than hardware prefetching alone.

Like the computational Roofline, the computational ceilings can come from an optimization manual [2], although it's easy to imagine collecting the necessary parameters from simple microbenchmarks. The memory ceilings require running experiments on each computer to determine the gap between them (see Appendix A.1). The good news is that like the Roofline, the ceilings only need be measured once per multicore computer.

Figure 2 adds ceilings to the Roofline model in Figure 1a: Figure 2a shows the computational ceilings and Figure 2b the memory bandwidth ceilings. Although the higher ceilings are not labeled with lower optimizations, they are implied: to break through a ceiling, you need to have already broken through all the ones below. Figure 2a shows the computational "ceilings" of 8.8 GFlops/sec if the floating-point operation mix is imbalanced and 2.2 GFlops/sec if the optimizations to increase ILP or SIMD are also missing. Figure 2b shows the memory bandwidth ceilings of 11 GBytes/sec without software prefetching, 4.8 GBytes/sec without memory affinity optimizations as well, and 2.7 GBytes/sec with only unit stride optimizations.

Figure 2c combines the other two figures into a single graph. The operational intensity of a kernel determines the optimization region, and thus which optimizations to try. The middle of Figure 2c shows that the computational optimizations and the memory bandwidth optimizations overlap. The colors were picked to highlight that overlap. For example, Kernel 2 falls in the blue trapezoid on the right, which suggests working only on the computational optimizations. If a kernel fell in the yellow triangle on the lower left, the model would suggest trying just memory optimizations. Kernel 1 falls in the green (= yellow + blue) parallelogram in the middle, which suggests trying both types of optimizations. Note that the Kernel 1 vertical lines falls below the floating-point imbalance optimization, so optimization 2 may be skipped.

The ceilings of the Roofline model suggest which optimizations to perform. The height of the gap between a ceiling and the next higher one is the potential reward for trying that optimization. Thus, Figure 2 suggests that optimization 1, which improves ILP/SIMD, has a large potential benefit for improving computation on that computer, and optimization 4, which improves memory affinity, has a large potential benefit for improving memory bandwidth on that computer.

The order of the ceilings suggest the optimization order, so we rank the ceilings from bottom to top: those most likely to be realized by a compiler or with little effort by a programmer are at the bottom and those that are difficult to be implemented by a programmer or inherently lacking in a kernel are at the top. The one quirk is floating-point balance, since the actual mix is dependent on the kernel. For most kernels, achieving parity between multiplies and additions is very difficult, but for a few, parity is natural. One example is sparse matrix-vector multiplication. For that domain, we would place floating-point mix as the lowest ceiling, since it is inherent. Like the 3Cs model, as long as the Roofline model delivers on insights, it need not be perfect.

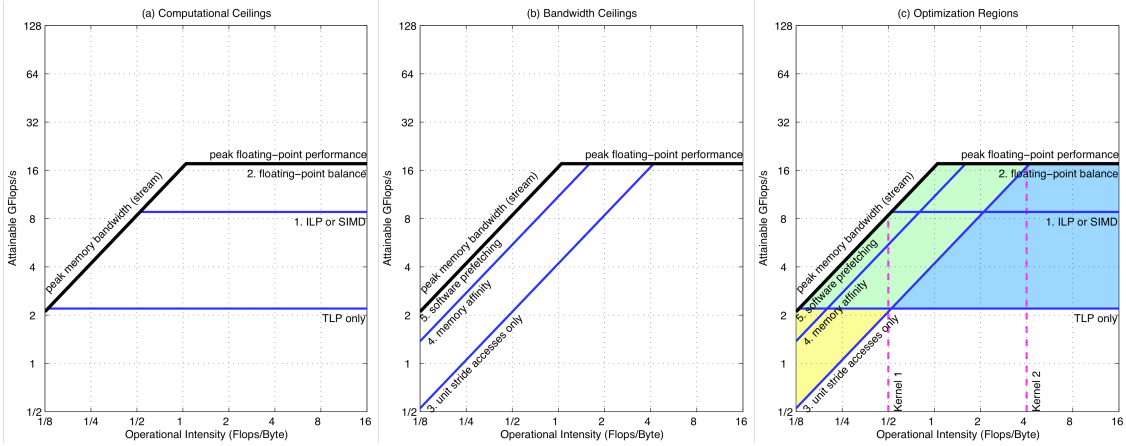

Figure 2. Roofline Model with Ceilings for Opteron X2.

性能通常还需要相等数量的同时浮点加法和乘法，因为许多计算机具有乘加指令，或者因为它们有相等数量的加法器和乘法器。

为了减少内存瓶颈，可以采取三种优化措施：

3. *Restructure loops for unit stride accesses*. 优化单位步幅内存访问会启用硬件预取，这显著提高了内存带宽。

4. *Ensure memory affinity*. 目前大多数微处理器在同一芯片上都包含一个内存控制器。如果系统有两个多核芯片，那么一些地址会指向一个多核芯片本地的DRAM，其余的必须通过芯片互连访问另一个芯片本地的DRAM。这种情况会降低性能。此优化将数据和负责该数据的线程分配给相同的内存-处理器对，以便处理器很少需要访问附加到其他芯片的内存。

5. *Use software prefetching*. 通常，最高性能需要保持许多内存操作处于进行中，这通过预取比等待程序实际请求数据更容易。在某些计算机上，软件预取提供的带宽比仅依赖硬件预取要高。

像计算 Roofline 一样，计算上限可以来自优化手册 [2]，尽管很容易想象从简单的微基准中收集必要的参数。内存上限需要在每台计算机上运行实验以确定它们之间的差距（见附录 A.1）。好消息是，像 Roofline 一样，上限只需在每台多核计算机上测量一次。

图2在图1a的Roofline模型中添加了上限：图2a显示了计算上限，图2b显示了内存带宽上限。尽管较高的上限没有标注较低的优化，但它们是隐含的：要突破一个上限，您需要已经突破所有下面的上限。图2a显示了8.8 GFlops/sec的计算“上限”，如果浮点操作的混合不平衡并且

如果缺少增加 ILP 或 SIMD 的优化，则为 2.2 GFlops/秒。图 2b 显示了在没有软件预取的情况下，内存带宽上限为 11 GBytes/秒，在没有内存亲和性优化的情况下为 4.8 GBytes/秒，仅使用单位步幅优化时为 2.7 GBytes/秒。

图2c将其他两个图合并为一个单一的图表。内核的操作强度决定了优化区域，从而决定尝试哪些优化。图2c的中间部分显示计算优化和内存带宽优化重叠。颜色的选择旨在突出这种重叠。例如，内核2位于右侧的蓝色梯形中，这表明只需关注计算优化。如果一个内核位于左下角的黄色三角形中，模型将建议仅尝试内存优化。内核1位于中间的绿色（=黄色+蓝色）平行四边形中，这表明应尝试两种类型的优化。请注意，内核1的垂直线位于浮点不平衡优化的下方，因此可能会跳过优化2。

Roofline模型的天花板指示了应该进行哪些优化。天花板与下一个更高天花板之间的间隙高度是尝试该优化的潜在回报。因此，图2表明，优化1（改善ILP/SIMD）对提高该计算机的计算有很大的潜在好处，而优化4（改善内存亲和性）对提高该计算机的内存带宽也有很大的潜在好处。

天花板的顺序暗示了优化的顺序，因此我们将天花板从下到上进行排序：那些最有可能被编译器实现或程序员以较小努力实现的位于底部，而那些程序员难以实现或本质上缺乏内核的位于顶部。一个特例是浮点平衡，因为实际的混合依赖于内核。对于大多数内核，实现乘法和加法之间的平衡是非常困难的，但对于少数内核，平衡是自然的。一个例子是稀疏矩阵-向量乘法。对于该领域，我们将浮点混合视为最低的天花板，因为这是内在的。像3Cs模型一样，只要Roofline模型提供洞察，它就不必是完美的。

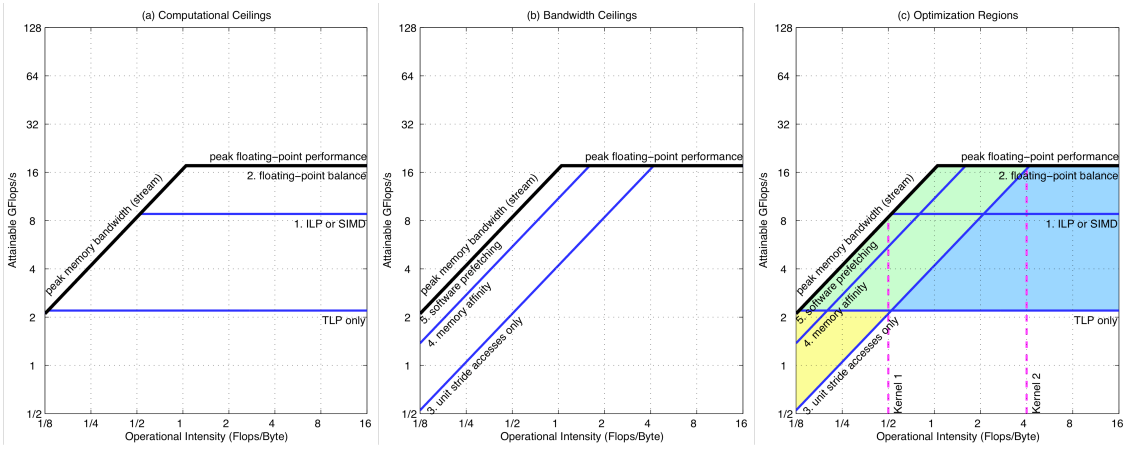

图 2. Opteron X2 的屋顶线模型及其上限。

5. Tying the 3Cs to Operational Intensity

Operational intensity tells us which ceilings to look at. Thus far, we have been assuming that the operational intensity is fixed, but that is not really the case. For example, there are kernels where the operational intensity increases with problem size, such as for Dense Matrix and FFT problems.

Clearly, caches affect the number of accesses that go to memory, so optimizations that improve cache performance increase operational intensity. Hence, we can connect the 3Cs model to the Roofline model. Compulsory misses set the minimum memory traffic and hence the highest possible operational intensity. Memory traffic from conflict and capacity misses can considerably lower the operational intensity of a kernel, so we should try to eliminate such misses.

For example, we can reduce traffic from conflict misses by padding arrays to change cache line addressing. A second example is that some computers have a no-allocate store instruction, so stores go directly to memory and do not affect the caches. This optimization prevents loading a cache block with data to be overwritten, thereby reducing memory traffic. It also prevents displacing useful items in the cache with data that will not be read thereby saving conflict misses.

This shift right of operational intensity could put a kernel in a different optimization region. The advice is generally to improve operational intensity of the kernel before other optimizations.

6. DEMONSTRATION OF THE MODEL

To demonstrate the utility of the model, we develop Roofline models for 4 recent multicore computers and then optimize 4 floating-point kernels. We then show that the ceilings and rooflines bound the achieved results for all computers and kernels.

6.1 Four Diverse Multicore Computers

Given the lack of conventional wisdom for multicore architecture, it's not surprising that there are as many different designs as there are chips. Table 1 lists the key characteristics of the four multicore computers of this section, which are all dual-socket systems.

The Intel Xeon uses relatively sophisticated processors, capable of executing two SIMD instructions per clock cycle that can each perform two double-precision floating-point operations. It is the only one of the four machines with a front side bus connecting to a common north bridge chip and memory controller. The other three have the memory controller on chip.

The Opteron X4 also uses sophisticated cores with high peak floating–point performance, but it is the only computer of the four with on-chip L3 caches. These two sockets communicate over separate, dedicated Hypertransport links, which makes it possible to build a "glueless" multi-chip system.

The Sun UltraSPARC T2+ uses relatively simple processors at a modest clock rate compared to the others, which allows it to have twice as many cores per chip. It is also highly multithreaded, with eight hardware-supported threads per core. It has the highest memory bandwidth of the four, for each chip has two dual-channel memory controllers that can drive four sets of DDR2/FBDIMMs.

The clock rate of IBM Cell QS20 is highest of the four multicores at 3.2 GHz. It is also most unusual. It is a heterogeneous design, with a relatively simple PowerPC core and with eight SPEs (Synergistic Processing Elements) that have their own unique SIMD-style instruction set. Each SPE also has its own local memory instead of a cache. An SPE must transfer data from main memory into the local memory to operate on it and then back to main memory when it is completed. It uses DMA, which has some similarity to software prefetching. The lack of caches means porting programs to Cell is more challenging.

Table 1. Characteristics of four recent multicores.

MPU Type	Intel Xeon (Clovertown, e5345)	AMD Opteron X4 (Barcelona, 2356)	Sun UltraSPARC T2+ (Niagara 2, 5120)	IBM Cell (QS20)
ISA	x86/64	x86/64	SPARC	Cell SPEs
Total Threads	8	8	128	16
Total Cores	8	8	16	16
Total Sockets	2	2	2	2
GHz	2.33	2.30	1.17	3.20
Peak GFlop/s	75	74	19	29
Peak DRAM GB/s	21.3r, 10.6w	2 x 10.6	2 x 21.3r, 2 x 10.6w	2 x 25.6
Stream GB/s	5.9	16.6	26.0	47.0
DRAM Type	FBDIMM	DDR2	FBDIMM	XDR

6.2 Four Diverse Floating-Point Kernels

Rather than pick programs from some standard parallel benchmark suite such as Parsec [5] or Splash-2 [30], we were inspired by the work of Phil Colella [10]. This expert in scientific computing has identified seven numerical methods that he believes will be important for science and engineering for at least the next decade. Because he picked seven, they have become known as the *Seven Dwarfs*. The dwarfs are specified at a high level of abstraction to allow reasoning about their behavior across a broad range of implementations. The widely read "Berkeley View" report [4] found that if the data types were changed from floating point to integer, those same dwarfs could also be found in many other programs. Note that the claim is *not* that the dwarfs are easy to parallelize. The claim is that they will be important to computing in most current and future applications, so designers are advised to make sure they run well on systems that they create, whether or not their creations are parallel.

One advantage of using these higher-level descriptions of programs is that we are not tied to code that may have been written originally to optimize an old computer to evaluate future systems. Another advantage of the restricted number is that we can create *autotuners* for each kernel that would search the space

5. 将3Cs与运营强度联系起来

操作强度告诉我们应该关注哪些上限。到目前为止，我们一直假设操作强度是固定的，但实际上并非如此。例如，有些内核的操作强度会随着问题规模的增加而增加，例如稠密矩阵和快速傅里叶变换（FFT）问题。

显然，缓存会影响访问内存的次数，因此，改善缓存性能的优化会增加操作强度。因此，我们可以将3Cs模型与Roofline模型联系起来。强制缺失设定了最小内存流量，从而设定了最高可能的操作强度。来自冲突和容量缺失的内存流量会显著降低内核的操作强度，因此我们应该尽量消除这些缺失。

例如，我们可以通过填充数组来改变缓存行地址，从而减少由于冲突未命中造成的流量。第二个例子是，一些计算机具有不分配存储指令，因此存储直接进入内存，不会影响缓存。这种优化防止了用将要被覆盖的数据加载缓存块，从而减少了内存流量。它还防止用不会被读取的数据替换缓存中的有用项，从而节省了冲突未命中。

这种操作强度的右移可能会将内核置于不同的优化区域。一般建议在进行其他优化之前，先提高内核的操作强度。

6. 模型演示

为了展示模型的实用性，我们为4台最近的多核计算机开发了Roofline模型，然后优化了4个浮点内核。我们接着展示了天花板和Roofline限制了所有计算机和内核的实际结果。

6.1 四种多样的多核计算机

鉴于对多核架构缺乏传统智慧，出现与芯片数量相同的不同设计并不令人惊讶。表1列出了本节四台多核计算机的关键特性，它们都是双插槽系统。

英特尔至强使用相对复杂的处理器，能够在每个时钟周期内执行两个SIMD指令，每个指令可以执行两个双精度浮点运算。它是四台机器中唯一一台具有连接到公共北桥芯片和内存控制器的前端总线。其他三台则将内存控制器集成在芯片上。

Opteron X4 也使用高峰浮点性能的复杂核心，但它是四个计算机中唯一具有片上 L3 缓存的。 这两个插槽通过独立的专用 Hypertransport 链接进行通信，这使得构建“无胶”多芯片系统成为可能。

Sun UltraSPARC T2+ 使用相对简单的处理器，时钟频率与其他处理器相比适中，这使得它每个芯片可以拥有两倍的核心数量。它还具有高度的多线程支持，每个核心支持八个硬件线程。它具有最高的四个芯片的内存带宽，每个芯片有两个双通道内存控制器，可以驱动四组DDR2/FBDIMM。

IBM Cell QS20 的时钟频率在四个多核中最高，为 3.2 GHz。它也是最不寻常的。它是一种异构设计，具有相对简单的 PowerPC 核心和八个 SPE（协同处理单元），这些 SPE 拥有自己独特的 SIMD 风格指令集。每个 SPE 还拥有自己的本地内存，而不是缓存。SPE 必须将数据从主内存传输到本地内存以进行操作，然后在完成后再返回主内存。它使用 DMA，这与软件预取有一些相似之处。缺乏缓存意味着将程序移植到 Cell 更具挑战性。

Table 1. Characteristics of four recent multicores.

	AMD Opteron X4 (2356 "Barcelona")	Intel Xeon (E5345 "Clovertown")	Sun UltraSPARC T2+ (5120 "Niagara2")	IBM QS20 Cell Blade
ISA	x86/64	x86/64	SPARC	Cell SPEs
Total Threads	8	8	128	16
Total Cores	8	8	16	16
Total Sockets	2	2	2	2
GHz	2.33	2.30	1.17	3.20
Peak GFlop/s	75	74	19	29
Peak DRAM GB/s	21.3r, 10.6w	2 x 10.6	2 x 21.3r, 2 x 10.6w	2 x 25.6
Stream GB/s	5.9	16.6	26.0	47.0
DRAM Type	FBDIMM	DDR2	FBDIMM	XDR

6.2 四种多样的浮点内核

与其从一些标准的并行基准套件中选择程序，例如 Parsec [5] 或 Splash-2 [30]，我们受到 Phil Colella [10] 的工作的启发。这位科学计算专家确定了七种他认为在未来十年内对科学和工程将至关重要的数值方法。因为他选择了七种，所以它们被称为 *Seven Dwarfs*。这些 dwarf 在高抽象层次上被指定，以便能够推理它们在广泛实现中的行为。广泛阅读的“伯克利视角”报告 [4] 发现，如果数据类型从浮点数更改为整数，这些相同的 dwarf 也可以在许多其他程序中找到。请注意，声明是 *not*，即这些 dwarf 易于并行化。声明是它们将在大多数当前和未来的应用中对计算至关重要，因此建议设计师确保它们在他们创建的系统上运行良好，无论他们的创作是否是并行的。

使用这些更高级的程序描述的一个优势是，我们不必依赖可能最初为优化旧计算机而编写的代码来评估未来的系统。另一个限制数量的优势是，我们可以为每个内核创建 *autotuners*，以搜索该空间。

of alternatives to produce the best code for that multicore computer, including extensive cache optimizations [13].

With that background, Table 2 lists the four kernels from the dwarfs that we use to demonstrate the Roofline Model on the four multicore computers of Table 1. The auto-tuning for this section is from [12], [25] and [26].

For these kernels, there is sufficient parallelism to utilize all the cores and threads and to keep them load balanced. (Appendix A.2 describes how to handle cases when load is not balanced.)

Table 2. Characteristics of four FP Kernels.

Name	Oper. Inten.	Description
SpMV [26]	0.17 to 0.25	Sparse Matrix-Vector multiply: y = A*x where A is a sparse matrix and x, y are dense vectors; multiplies and adds equal.
LBMHD [25]	0.70 to 1.07	Lattice-Boltzmann Magnetohydro-dynamics is a structured grid code with a series of time steps.
Stencil [12]	0.33 to 0.50	A multigrid kernel that updates 7 nearby points in a 3-D stencil for a 256^3 problem
3-D FFT	1.09 to 1.64	Three-Dimensional Fast Fourier Transform (2 sizes: 128^3 and 512^3).

6.3 Roofline Models and Results

Figure 3 shows the Roofline models for Xeon, X4, and Cell. The pink vertical dashed lines show the operational intensity and the red X marks performance achieved for that kernel. As mentioned above, adds and multiplies are naturally equal in SpMV, so balance is easy for this kernel but hard for the others. Hence, there are two graphs per computer in Figure 3: the left graphs have multiply-add balance as the top ceiling for LBMHD, Stencil, and 3-D FFT, and those on the right have multiply-add as the bottom ceiling for SpMV. Since the T2+ does not have a fused multiply-add instruction nor can it simultaneously issue multiplies and adds, Figure 4 shows a single roofline for the four kernels for T2+ without the multiply-add balance ceiling.

The Intel Xeon has the highest peak double precision performance of the four multicores. However, the Roofline model in Figure 3a shows that this can be achieved only with operational intensities of at least 6.7; started alternatively, balance requires 55 floating-point operations for every double precision operand (8 bytes) going to DRAM. This high ratio is due in part to the limitation of the front side bus, which also carries coherency traffic that can consume half the bus bandwidth. Intel includes a *snoop filter* to prevent unnecessary coherency traffic on the bus. If the working set is small enough for the hardware to filter, the snoop filter nearly doubles the delivered memory bandwidth.

The Opteron X4 has a memory controller on chip, its own path to 667 MHz DDR2 DRAM, and separate paths for coherency. Figure 3 shows that the ridge point in the Roofline model is to the left of the Xeon, at an operational intensity of 4.4 Flops per byte. The Sun T2+ has the highest memory bandwidth so the ridge point is an exceptionally low operational intensity of just 0.33 Flops per byte. It keeps multiple memory transfers in flight by using many threads. The IBM Cell ridge point of operational intensity is 0.65.

6.3.1 Sparse Matrix-Vector Multiplication

The first example kernel of the sparse matrix computational dwarf is Sparse Matrix-Vector multiply (SpMV). The computation is $y = A*x$ where A is a sparse matrix and x and y are dense vectors. SpMV is popular in scientific computing, economic modeling, and information retrieval. Alas, conventional implementations often run at less than 10% of peak floating-point performance in uniprocessors. One reason is the irregular accesses to memory, which you might expect from sparse matrices. The operational intensity varies from 0.17 before a register blocking optimization to 0.25 Flops per byte afterwards [29]. (See Appendix A.1.)

Given that the operational intensity of SpMV was below the ridge point of all four multicores in Figure 3, most of the optimizations involved the memory system. Table 3 summarizes the optimizations used by SpMV and the rest of the kernels. Many are associated with the ceilings in Figure 3, and the height of the ceilings suggests the potential benefit of these optimizations.

6.3.2 Lattice-Boltzmann Magnetohydrodynamics

Like SpMV, LBMHD tends to get a small fraction of peak performance on uniprocessors because of the complexity of the data structures and the irregularity of memory access patterns. The Flops to byte ratio is 0.70 versus 0.25 or less in SpMV. By using the no-allocate store optimization, the LBMHD intensity rises to 1.07. Both x86 multicores offer this cache optimization, and Cell does not have this problem since it uses DMA. Hence, T2+ is the only one with the lower intensity of 0.70.

Figures 3 and 4 show that the operational intensity of LBMHD is high enough that both computational and memory bandwidth optimizations make sense on all multicores but the T2+, whose Roofline ridge point is below that of LBMHD. The T2+ reaches its performance ceiling using only the computational optimizations.

6.3.3 Stencil

In general, a stencil on a structure grid is defined as a function that updates a point based on the values of its neighbors. The stencil structure remains constant as it moves from one point in space to the next. For this work, we use the stencil derived from the explicit heat equation PDE on a uniform 256^3 3-D grid [12]. The neighbors for this stencil are the nearest 6 points along each axis as well as the center point itself. This stencil will do 8 floating-point operations for every 24 bytes of compulsory memory traffic on write-allocate architectures, yielding an operational intensity of 0.33.

6.3.4 3-D FFT

This fast Fourier transform is the classic divide and conquer algorithm that recursively breaks down a discrete Fourier transform into many smaller ones. The FFT is ubiquitous in many domains, such as image processing and data compression. An efficient approach for 3-D FFT is to perform 1-D transforms along each dimension to maintain unit-stride accesses. We computed the 1-D FFTs on Xeon, X4, and T2+ using an autotuned library (FFTW) [15]. For Cell, we implemented a radix-2 FFT.

为该多核计算机生成最佳代码的替代方案，包括广泛的缓存优化 [13]。

在这样的背景下，表2列出了我们用来在表1的四个多核计算机上演示屋顶线模型的四个来自矮人的内核。本节的自动调优来自[12]、[25]和[26]。

对于这些内核，有足够的并行性来利用所有的核心和线程，并保持负载平衡。（附录 A.2 描述了如何处理负载不平衡的情况。）

Table 2. Characteristics of four FP Kernels.

Name	Oper. Inten.	Description
SpMV [26]	0.17 to 0.25	Sparse Matrix-Vector multiply: y = A*x where A is a sparse matrix and x, y are dense vectors; multiplies and adds equal.
LBMHD [25]	0.70 to 1.07	Lattice-Boltzmann Magnetohydro-dynamics is a structured grid code with a series of time steps.
Stencil [12]	0.33 to 0.50	A multigrid kernel that updates 7 nearby points in a 3-D stencil for a 256^3 problem
3-D FFT	1.09 to 1.64	Three-Dimensional Fast Fourier Transform (2 sizes: 128^3 and 512^3).

6.3 屋顶线模型和结果

图3显示了Xeon、X4和Cell的Roofline模型。粉色的垂直虚线表示操作强度，红色X标记表示该内核实现的性能。如上所述，SpMV中的加法和乘法自然是相等的，因此对于这个内核来说平衡很容易，但对于其他内核则很困难。因此，图3中每台计算机有两个图：左侧的图以乘加平衡作为LBMHD、Stencil和3-D FFT的上限，而右侧的图则以乘加作为SpMV的下限。由于T2+没有融合乘加指令，也不能同时发出乘法和加法，图4显示了四个内核的T2+的单一Roofline，没有乘加平衡上限。

英特尔至强处理器在四个多核中具有最高的峰值双精度性能。然而，图3a中的Roofline模型显示，这只能在操作强度至少为6.7的情况下实现；反之，平衡要求每个双精度操作数（8字节）在进入DRAM时需要55次浮点操作。这个高比例部分是由于前端总线的限制，该总线还承载着可能消耗一半总线带宽的连贯性流量。英特尔包括一个*snoop filter*以防止总线上不必要的连贯性流量。如果工作集足够小以便硬件进行过滤，嗅探过滤器几乎可以使交付的内存带宽翻倍。

Opteron X4 在芯片上有一个内存控制器，拥有自己通往 667 MHz DDR2 DRAM 的路径，并且有单独的路径用于一致性。图 3 显示，Roofline 模型中的岭点位于 Xeon 的左侧，操作强度为每字节 4.4 Flops。Sun T2+ 拥有最高的内存带宽，因此岭点的操作强度异常低，仅为每字节 0.33 Flops。它通过使用多个线程保持多个内存传输处于飞行状态。IBM Cell 的操作强度岭点为 0.65。

6.3.1 Sparse Matrix-Vector Multiplication

稀疏矩阵计算矮人第一个示例内核是稀疏矩阵-向量乘法（SpMV）。计算为 $y = A*x$，其中 A 是一个稀疏矩阵，而 x 和 y 是密集向量。SpMV 在科学计算、经济建模和信息检索中很受欢迎。可惜，常规实现通常在单处理器上运行的浮点性能不到峰值的 10%。一个原因是对内存的不规则访问，这在稀疏矩阵中是可以预期的。操作强度从注册阻塞优化前的 0.17 变化到之后的每字节 0.25 Flops [29]。（见附录 A.1。）

考虑到SpMV的操作强度低于图3中所有四个多核的峰值点，大多数优化涉及内存系统。表3总结了SpMV和其他内核使用的优化。许多与图3中的上限相关，上限的高度暗示了这些优化的潜在好处。

6.3.2 Lattice-Boltzmann Magnetohydrodynamics

像 SpMV 一样，LBMHD 在单处理器上由于数据结构的复杂性和内存访问模式的不规则性，往往只能获得小部分的峰值性能。Flops 与字节的比率为 0.70，而 SpMV 则为 0.25 或更低。通过使用无分配存储优化，LBMHD 的强度上升到 1.07。两个 x86 多核都提供这种缓存优化，而 Cell 由于使用 DMA，因此没有这个问题。因此，T2+ 是唯一一个强度较低的 0.70。

图3和图4显示，LBMHD的操作强度足够高，以至于在所有多核上进行计算和内存带宽优化都是有意义的，但T2+的Roofline边界点低于LBMHD。T2+仅通过计算优化达到了其性能上限。

6.3.3 Stencil

一般来说，结构网格上的模板被定义为一个函数，该函数根据其邻居的值更新一个点。模板结构在从空间中的一个点移动到下一个点时保持不变。对于这项工作，我们使用从均匀的256^3 3-D网格上的显式热方程PDE导出的模板[12]。该模板的邻居是沿每个轴的最近6个点以及中心点本身。这个模板在写分配架构上每24字节的强制内存流量将执行8次浮点操作，产生0.33的操作强度。

6.3.4 3-D FFT

这个快速傅里叶变换是经典的分治算法，它递归地将离散傅里叶变换分解为许多更小的变换。FFT在许多领域中无处不在，例如图像处理和数据压缩。3-D FFT的一个高效方法是在每个维度上执行1-D变换，以保持单位步幅访问。我们在Xeon、X4和T2+上使用自调优库（FFTW）计算了1-D FFT。对于Cell，我们实现了一个基数-2 FFT。

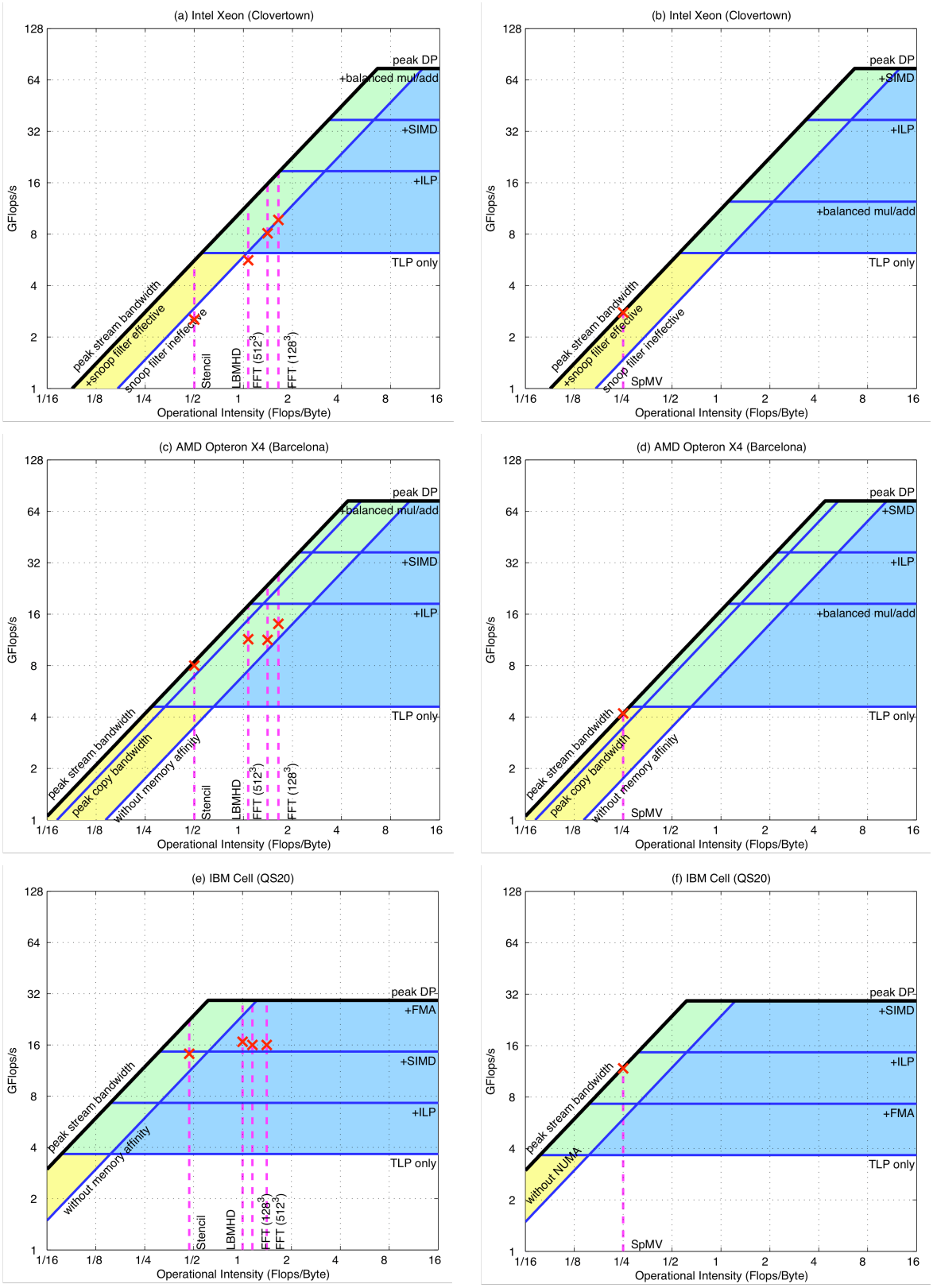

Figure 3. Roofline Model for Intel Xeon, AMD Opteron X4, and IBM Cell (see Table 1).

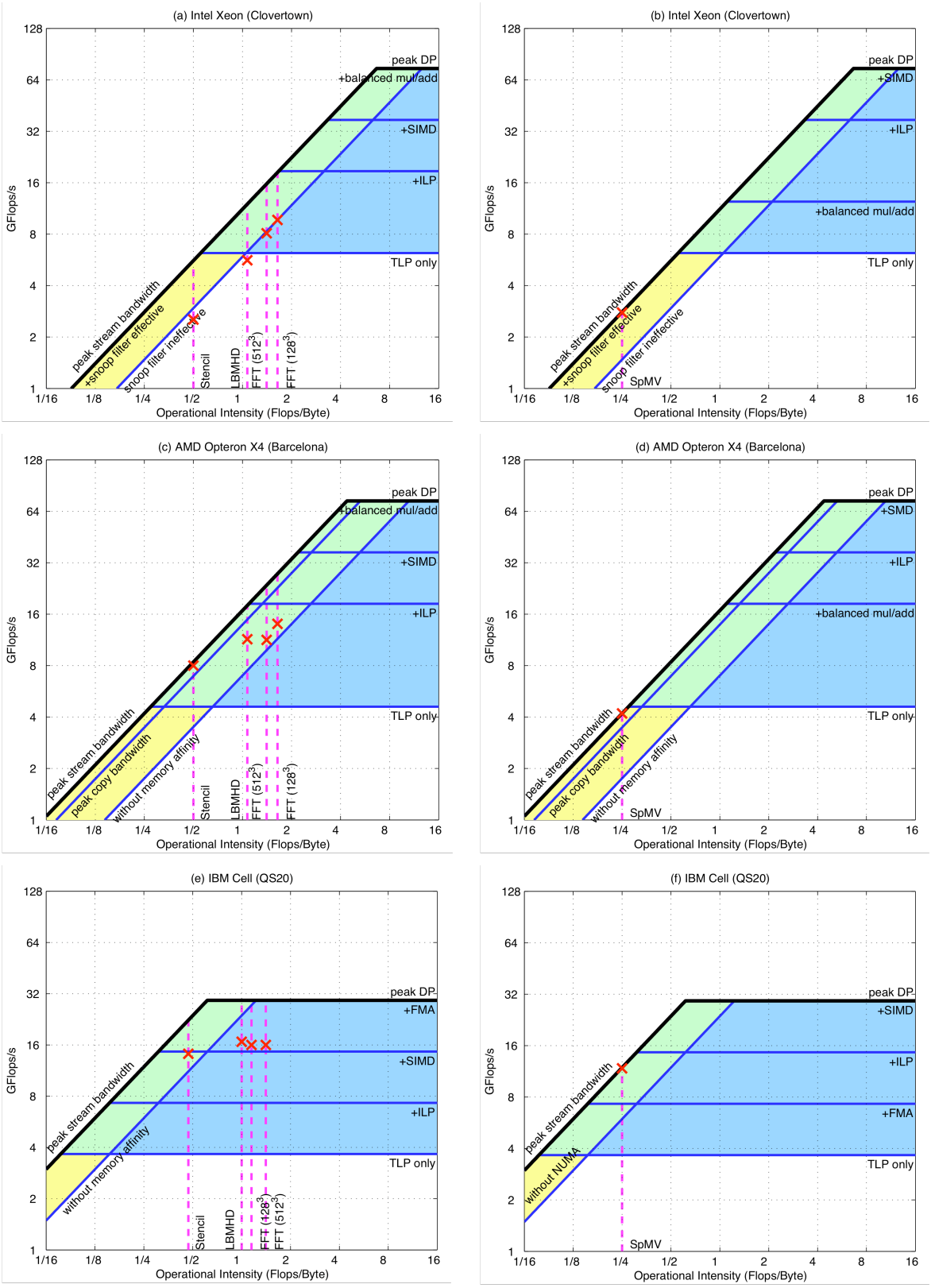

图 3. Intel Xeon、AMD Opteron X4 和 IBM Cell 的 Roofline 模型（见表 1）。

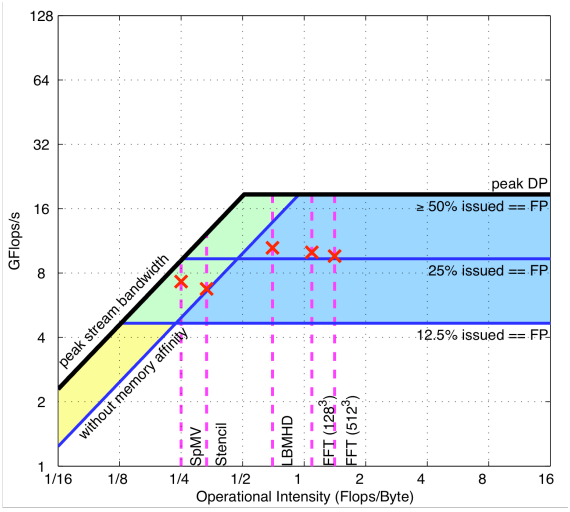

Figure 4. Roofline Model for Sun UltraSPARC T2+.

Table 3. Kernel Optimizations [12], [26] [25].
Memory Affinity. Reduce accesses to DRAM memory attached to the other socket.
Long unit-stride accesses. Change loop structures to generate long unit-stride accesses to engage the prefetchers. Also reduces TLB misses.
Software Prefetching. To get the most out of the memory systems, both software and hardware prefetching were used.
Reduce conflict misses. Pad arrays to improve cache-hit rates.
Unroll and Reorder Loops. To expose sufficient parallelism and improve cache utilization, unroll and reorder loops to group statements with similar addresses; improves code quality, reduces register pressure, and facilitates SIMD.
"SIMD-ize" the code. The x86 compilers didn't generate good SSE code, so made a code generator to produce SSE intrinsics.
Compress Data Structures (SpMV only). Since bandwidth limits performance, use smaller data structures: 16-bit vs. 32-bit index and smaller representations of non-zero subblocks [24].

FFT differs from the three kernels above in that its operational intensity is a function of problem size. For the 128^3- and 512^3-point transforms we examine, the operational intensities are 1.09 and 1.41, respectively. (Cell's 1 GB main memory is too small to hold 512^3 points, so we estimate this result.) On Xeon and X4, an entire 128x128 plane fits in cache, increasing temporal locality and improving the intensity to 1.64 for the 128^3-point transform.

6.3.5 *Productivity vs. Performance*

In addition to performance, another important issue for the parallel computing revolution is productivity, or the programming difficulty of achieving good performance [4]. One question is whether a low ridge point gives insight into productivity.

The Sun T2+, with the lowest ridge point, was easiest to program, due to its large memory bandwidth and its easy-to-understand cores. The advice for these kernels on T2+ is simply to try to get good performing code from the compiler and then use as many threads as possible. The downside was that the L2 cache was only 16-way set associative, which can lead to conflict misses when 64 threads access the cache, as it did for Stencil.

In contrast, the computer with the highest ridge point had the lowest unoptimized performance. The Intel Xeon was difficult because it was hard to understand the memory behavior of the dual front side buses, hard to understand how hardware prefetching worked, and because of the difficulty of getting good SIMD code from the compiler. The C code for it and for the Opteron X4 are liberally sprinkled with intrinsic statements involving SIMD instructions to get good performance. With a ridge point close to the Xeon, the Opteron X4 was about as much effort, since the Opteron X4 benefited from the most types of optimizations. However, the memory behavior of the Opteron X4 was easier to understand than that of the Xeon.

The IBM Cell, with a ridge point almost as low as the Sun T2+, provided two types of challenges. First, it was awkward to compile for the SIMD instructions of Cell's SPE, so at times we needed to help the compiler by inserting intrinsic statements with assembly language instructions into the C code. This comment reflects the immaturity of the IBM compiler as well as the difficulty of compiling for these SIMD instructions. Second, the memory system was more challenging. Since each SPE has local memory in a separate address space, we could not simply port the code and start running on the SPE. We needed to change the program to issue DMA commands to transfer data back and forth between local store and memory. The good news is that DMA played the role of software prefetch in caches. DMA for a local store is easier to program, to achieve good memory performance, and to overlap with computation than prefetching to caches.

6.3.6 *Summary of Roofline Model Demonstration*

To demonstrate the utility of the Roofline Model, Table 4 shows upper and lower ceilings and the GFlops/s and GByte/s per kernel-computer pair; recall that operational intensity is the ratio between the two rates. The ceilings listed are the ceilings that sandwich the actual performance. All 16 cases validate this bound and bottleneck model since the upper and lower ceilings of Roofline bound performance and the kernels were optimized as the lower ceilings suggest. The metric that limits performance is in bold: 15 of 16 ceilings are memory bound for Xeon and X4 while it's almost evenly split for T2+ and Cell. For FFT, interestingly, the surrounding ceilings are memory bound for Xeon and X4 but compute bound for T2+ and Cell.

7. FALLACIES ABOUT ROOFLINE

We have presented this material in several venues, so there are some common questions that arise that we answer here.

Fallacy: The model does not take into account all features of modern processors, such as caches or prefetching.

The definition of operational intensity in this paper does indeed factor in caches: memory accesses are measured between the caches and memory, not between the processor and caches.

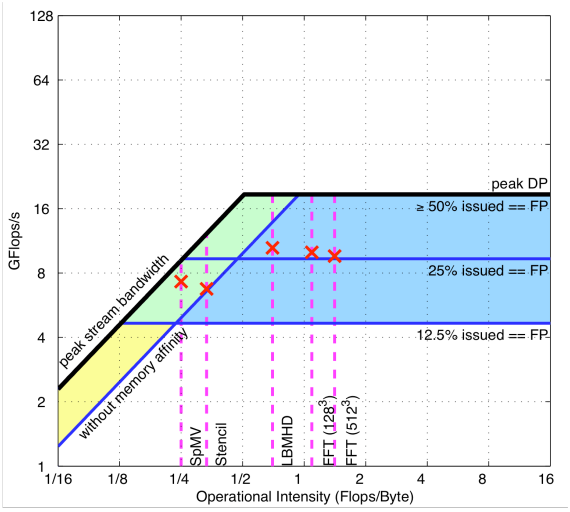

图 4. Sun UltraSPARC T2+ 的屋顶线模型。

Table 3. Kernel Optimizations [12], [26] [25].
Memory Affinity. Reduce accesses to DRAM memory attached to the other socket.
Long unit-stride accesses. Change loop structures to generate long unit-stride accesses to engage the prefetchers. Also reduces TLB misses.
Software Prefetching. To get the most out of the memory systems, both software and hardware prefetching were used.
Reduce conflict misses. Pad arrays to improve cache-hit rates.
Unroll and Reorder Loops. To expose sufficient parallelism and improve cache utilization, unroll and reorder loops to group statements with similar addresses; improves code quality, reduces register pressure, and facilitates SIMD.
"SIMD-ize" the code. The x86 compilers didn't generate good SSE code, so made a code generator to produce SSE intrinsics.
Compress Data Structures (SpMV only). Since bandwidth limits performance, use smaller data structures: 16-bit vs. 32-bit index and smaller representations of non-zero subblocks [24].

FFT 与上述三个内核不同，因为它的操作强度是问题大小的函数。对于我们检查的 128^3 和 512^3 点变换，操作强度分别为 1.09 和 1.41。（Cell 的 1 GB 主内存太小，无法容纳 512^3 点，因此我们估计这个结果。）在 Xeon 和 X4 上，整个 128x128 平面适合缓存，增加了时间局部性，并将 128^3 点变换的强度提高到 1.64。

6.3.5 *Productivity vs. Performance*

除了性能，平行计算革命的另一个重要问题是生产力，即实现良好性能的编程难度 [4]。一个问题是，低岭点是否能提供对生产力的洞察。

太阳 T2+，具有最低的脊点，因其大内存带宽和易于理解的核心而最容易编程。对于 T2+ 上的这些内核的建议就是尽量尝试获取良好的编译器性能代码，然后尽可能使用多个线程。缺点是L2缓存仅为16路组相联，当64个线程访问缓存时可能会导致冲突缺失，就像Stencil那样。

相对而言，具有最高峰值的计算机的未优化性能最低。英特尔至强处理器很难，因为很难理解双前端总线的内存行为，难以理解硬件预取的工作原理，并且从编译器获得良好的SIMD代码也很困难。它和Opteron X4的C代码中大量使用了涉及SIMD指令的内在语句，以获得良好的性能。由于与至强处理器的峰值接近，Opteron X4的工作量大致相当，因为Opteron X4受益于最多类型的优化。然而，Opteron X4的内存行为比至强处理器更容易理解。

IBM Cell 的峰值几乎与 Sun T2+ 一样低，提供了两种类型的挑战。首先，为 Cell 的 SPE 的 SIMD 指令编译是很麻烦的，因此有时我们需要通过在 C 代码中插入带有汇编语言指令的内联语句来帮助编译器。这个评论反映了 IBM 编译器的不成熟以及为这些 SIMD 指令编译的困难。其次，内存系统更具挑战性。由于每个 SPE 在一个单独的地址空间中具有本地内存，我们不能简单地移植代码并开始在 SPE 上运行。我们需要更改程序以发出 DMA 命令，在本地存储和内存之间来回传输数据。好消息是，DMA 在缓存中充当了软件预取的角色。与预取到缓存相比，本地存储的 DMA 更容易编程，以实现良好的内存性能，并与计算重叠。

6.3.6 *Summary of Roofline Model Demonstration*

为了展示屋顶线模型的实用性，表4显示了每对内核-计算机的上限和下限，以及每秒的GFlops和每秒的GByte；请记住，操作强度是这两个速率之间的比率。列出的上限是夹住实际性能的上限。所有16个案例验证了这个界限和瓶颈模型，因为屋顶线的上限和下限界定了性能，并且内核的优化符合下限的建议。限制性能的指标以粗体显示：16个上限中有15个对于Xeon和X4是内存绑定，而对于T2+和Cell则几乎是均匀分配的。有趣的是，对于FFT，周围的上限对于Xeon和X4是内存绑定，但对于T2+和Cell则是计算绑定。

7. 关于屋线的谬论

我们在多个场合展示了这些材料，因此出现了一些常见问题，我们在这里进行解答。

Fallacy: The model does not take into account all features of modern processors, such as caches or prefetching.

本文中操作强度的定义确实考虑了缓存：内存访问是在缓存和内存之间测量的，而不是在处理器和缓存之间。

Table 4. Achieved Performance and Nearest Roofline Ceilings, with Metric Limiting Performance in Bold (3-D FFT is 128^3).

	Kernel	*Upper Ceiling*			*Achieved Performance*			*Lower Ceiling*		
		Type	*Name*	*Value*	*Compute*	*Memory*	*O.I.*	*Type*	*Name*	*Value*
Intel Xeon	SpMV	Memory	Stream BW	11.2 GByte/s	2.8 GFlop/s	**11.1 GB/s**	0.25	Memory	Snoop filter	5.9 GByte/s
	LBMHD	Memory	Snoop filter	5.9 GByte/s	5.6 GFlop/s	**5.3 GB/s**	1.07	Memory	(none)	0.0 GByte/s
	Stencil	Memory	Snoop filter	5.9 GByte/s	2.5 GFlop/s	**5.1 GB/s**	0.50	Memory	(none)	0.0 GByte/s
	3-D FFT	Memory	Snoop filter	5.9 GByte/s	9.7 GFlop/s	**5.9 GB/s**	1.64	Compute	TLP only	6.2 GFlop/s
AMD X4	SpMV	Memory	Stream BW	17.6 GByte/s	4.2 GFlop/s	**16.8 GB/s**	0.25	Memory	Copy BW	13.9 GByte/s
	LBMHD	Memory	Copy BW	13.9 GByte/s	11.4 GFlop/s	**10.7 GB/s**	1.07	Memory	No Affinity	7.0 GByte/s
	Stencil	Memory	Stream BW	17.6 GByte/s	8.0 GFlop/s	**16.0 GB/s**	0.50	Memory	Copy BW	13.9 GByte/s
	3-D FFT	Memory	Copy BW	13.9 GByte/s	14.0 GFlop/s	**8.6 GB/s**	1.64	Memory	No Affinity	7.0 GByte/s
Sun T2+	SpMV	Memory	Stream BW	36.7 GByte/s	7.3 GFlop/s	**29.1 GB/s**	0.25	Memory	No Affinity	19.8 GByte/s
	LBMHD	Memory	No Affinity	19.8 GByte/s	10.5 GFlop/s	**15.0 GB/s**	0.70	Compute	25% issued FP	9.3 GFlop/s
	Stencil	Compute	25% issued FP	9.3 GFlop/s	**6.8 GFlop/s**	20.3 GB/s	0.33	Memory	No Affinity	19.8 GByte/s
	3-D FFT	Compute	Peak DP	19.8 GFlop/s	**9.2 GFlop/s**	10.0 GB/s	1.09	Compute	25% issued FP	9.3 GFlop/s
IBM Cell	SpMV	Memory	Stream BW	47.6 GByte/s	11.8 GFlop/s	**47.1 GB/s**	0.25	Memory	FMA	7.3 GFlop/s
	LBMHD	Memory	No Affinity	23.8 GByte/s	16.7 GFlop/s	**15.6 GB/s**	1.07	Memory	Without FMA	14.6 GFlop/s
	Stencil	Compute	Without FMA	14.6 GFlop/s	**14.2 GFlop/s**	30.2 GB/s	0.47	Memory	No Affinity	23.8 GByte/s
	3-D FFT	Compute	Peak DP	29.3 GFlop/s	**15.7 GFlop/s**	14.4 GB/s	1.09	Compute	SIMD	14.6 GFlop/s

Section 2 shows that the memory bandwidth measures of the computer *do* include prefetching and any other optimization that can improve memory performance such as blocking. Similarly, some of the optimizations in Table 3 explicitly involve memory. Moreover, Section 5 demonstrates their effect on increasing operational intensity by reducing capacity and conflict misses.

Fallacy: Doubling cache size will increase operational intensity.

Autotuning three of the four kernels gets very close to the compulsory memory traffic; in fact, the resultant working set is sometimes only a small fraction of the cache. Increasing cache size helps only with capacity misses and possibly conflict misses, so a larger cache can have no effect on the operational intensity for those three kernels. For 128^3 3-D FFT, however, a large cache can capture a whole plane of a 3-D cube, which improves operational intensity by reducing capacity and conflict misses.

Fallacy: The model doesn't account for the long memory latency.

The ceilings for no software prefetching in Figures 3 and 4 are at lower memory bandwidth precisely because they cannot hide the long memory latency.

Fallacy: The model ignores integer units in floating-point programs, which can limit performance.

For the examples in this paper, the amount of integer code and the integer performance can affect performance. For example, the Sun UltraSPARC T2+ fetches two instructions per core per clock cycle, and it doesn't have the SIMD instructions of the x86 that can operate on two double-precision floating-point operands at a time. Relative to others, T2+ executes more integer instructions and executes them at a lower rate, which hurts overall performance.

Fallacy: The model has nothing to do with multicore.

Little's Law [21][20][17] dictates that to really push the limits of the memory system, considerable concurrency is necessary. That concurrency is more easily satisfied in a multicore than in a uniprocessor. While the bandwidth orientation of the Roofline model certainly works for uniprocessors, it is even more helpful for multicores.

Fallacy: You need to recalculate the model for every kernel.

The Roofline need to be calculated for given performance metrics and computer just once, and then guide the design for any program for which that metric is the critical performance metric. The examples in this paper used floating-point operations and memory traffic. The ceilings are measured once, but they can be reordered depending whether the multiplies and adds are naturally balanced or not in the kernel (see Section 4).

Note that the heights of the ceilings in this paper document the maximum potential gain of a code performing this optimization. An interesting future direction is to use performance counters to adjust the height of the ceilings and the order of the ceilings for a particular kernel to show the actual benefits of each optimization and the recommended order to try them (see Appendix A.3).

Fallacy: The model is limited to easily optimized kernels that never hit in the cache.

First, these kernels do hit in the cache. For example, the cache-hit rates of our three multicores with on-chip caches are at least 94% for stencil and 98% for FFT. Second, if the dwarfs were easy to optimize, that would bode well for the future of multicores. Our experience, however, is that it was not easy to create the fastest version of these numerical methods on the divergent multicore architectures presented here. Indeed, three of the results were considered significant enough to be accepted for publication at major conferences [12][25][26].

Fallacy: The model is limited to floating-point programs.

Our focus in this paper has also been on floating-point programs, so the two axes of the model are floating-point operations per second and the floating-point operational intensity of accesses to main memory. However, we believe the Roofline model can work for other kernels where the performance was a function of different performance metrics.

A concrete example is the transpose phase of 3-D FFT, which does no floating-point operations at all. Figure 5 shows a Roofline model for just this phase on Cell, with exchanges replacing Flops in the model. One exchange involves reading and writing 16 bytes, so its operational intensity is 1/32. Despite the

Table 4. Achieved Performance and Nearest Roofline Ceilings, with Metric Limiting Performance in Bold (3-D FFT is 128^3).

	Kernel	*Upper Ceiling*			*Achieved Performance*			*Lower Ceiling*		
		Type	*Name*	*Value*	*Compute*	*Memory*	*O.I.*	*Type*	*Name*	*Value*
Intel Xeon	SpMV	Memory	Stream BW	11.2 GByte/s	2.8 GFlop/s	**11.1 GB/s**	0.25	Memory	Snoop filter	5.9 GByte/s
	LBMHD	Memory	Snoop filter	5.9 GByte/s	5.6 GFlop/s	**5.3 GB/s**	1.07	Memory	(none)	0.0 GByte/s
	Stencil	Memory	Snoop filter	5.9 GByte/s	2.5 GFlop/s	**5.1 GB/s**	0.50	Memory	(none)	0.0 GByte/s
	3-D FFT	Memory	Snoop filter	5.9 GByte/s	9.7 GFlop/s	**5.9 GB/s**	1.64	Compute	TLP only	6.2 GFlop/s
AMD X4	SpMV	Memory	Stream BW	17.6 GByte/s	4.2 GFlop/s	**16.8 GB/s**	0.25	Memory	Copy BW	13.9 GByte/s
	LBMHD	Memory	Copy BW	13.9 GByte/s	11.4 GFlop/s	**10.7 GB/s**	1.07	Memory	No Affinity	7.0 GByte/s
	Stencil	Memory	Stream BW	17.6 GByte/s	8.0 GFlop/s	**16.0 GB/s**	0.50	Memory	Copy BW	13.9 GByte/s
	3-D FFT	Memory	Copy BW	13.9 GByte/s	14.0 GFlop/s	**8.6 GB/s**	1.64	Memory	No Affinity	7.0 GByte/s
Sun T2+	SpMV	Memory	Stream BW	36.7 GByte/s	7.3 GFlop/s	**29.1 GB/s**	0.25	Memory	No Affinity	19.8 GByte/s
	LBMHD	Memory	No Affinity	19.8 GByte/s	10.5 GFlop/s	**15.0 GB/s**	0.70	Compute	25% issued FP	9.3 GFlop/s
	Stencil	Compute	25% issued FP	9.3 GFlop/s	**6.8 GFlop/s**	20.3 GB/s	0.33	Memory	No Affinity	19.8 GByte/s
	3-D FFT	Compute	Peak DP	19.8 GFlop/s	**9.2 GFlop/s**	10.0 GB/s	1.09	Compute	25% issued FP	9.3 GFlop/s
IBM Cell	SpMV	Memory	Stream BW	47.6 GByte/s	11.8 GFlop/s	**47.1 GB/s**	0.25	Memory	FMA	7.3 GFlop/s
	LBMHD	Memory	No Affinity	23.8 GByte/s	16.7 GFlop/s	**15.6 GB/s**	1.07	Memory	Without FMA	14.6 GFlop/s
	Stencil	Compute	Without FMA	14.6 GFlop/s	**14.2 GFlop/s**	30.2 GB/s	0.47	Memory	No Affinity	23.8 GByte/s
	3-D FFT	Compute	Peak DP	29.3 GFlop/s	**15.7 GFlop/s**	14.4 GB/s	1.09	Compute	SIMD	14.6 GFlop/s

第2节显示，计算机*do*的内存带宽测量包括预取和任何其他可以提高内存性能的优化，例如分块。类似地，表3中的一些优化明确涉及内存。

此外，第5节展示了通过减少容量和冲突失误来提高操作强度的效果。

Fallacy: Doubling cache size will increase operational intensity.

自动调优四个内核中的三个非常接近强制内存流量；事实上，结果工作集有时仅是缓存的一小部分。增加缓存大小仅对容量未命中和可能的冲突未命中有帮助，因此更大的缓存对这三个内核的操作强度可能没有影响。然而，对于128^3 3-D FFT，较大的缓存可以捕获3-D立方体的整个平面，从而通过减少容量和冲突未命中来提高操作强度。

Fallacy: The model doesn't account for the long memory latency.

在图3和图4中，未进行软件预取的上限处于较低的内存带宽，正是因为它们无法隐藏较长的内存延迟。

Fallacy: The model ignores integer units in floating-point programs, which can limit performance.

在本文的示例中，整数代码的数量和整数性能可以影响性能。例如，Sun UltraSPARC T2+ 每个核心每个时钟周期获取两个指令，并且它没有 x86 的 SIMD 指令，后者可以同时对两个双精度浮点操作数进行操作。相对于其他处理器，T2+ 执行更多的整数指令，并且以较低的速率执行它们，这会影响整体性能。

Fallacy: The model has nothing to do with multicore.

小法则 [21][20][17] 规定，要真正推动内存系统的极限，需要相当大的并发性。在多核系统中，这种并发性比在单处理器中更容易满足。虽然 Roofline 模型的带宽导向确实适用于单处理器，但对于多核系统来说，它更为有用。

Fallacy: You need to recalculate the model for every kernel.

屋顶线只需针对给定的性能指标和计算机计算一次，然后指导任何程序的设计，该程序的关键性能指标就是该指标。本文中的示例使用了浮点运算和内存流量。天花板只需测量一次，但可以根据内核中乘法和加法是否自然平衡进行重新排序（见第4节）。

请注意，本文档中的天花板高度记录了执行此优化的代码的最大潜在增益。一个有趣的未来方向是使用性能计数器来调整天花板的高度和特定内核的天花板顺序，以展示每个优化的实际好处以及推荐的尝试顺序（见附录 A.3）。

Fallacy: The model is limited to easily optimized kernels that never hit in the cache.

首先，这些内核确实在缓存中命中。例如，我们的三款具有片上缓存的多核处理器的缓存命中率对于模板计算至少为94%，对于快速傅里叶变换（FFT）为98%。其次，如果这些小型程序容易优化，那将对多核处理器的未来是个好兆头。然而，我们的经验是，在这里展示的不同多核架构上，创建这些数值方法的最快版本并不容易。实际上，三个结果被认为足够重要，以至于被接受在主要会议上发表[12][25][26]。

Fallacy: The model is limited to floating-point programs.

我们在本文中的重点也放在了浮点程序上，因此模型的两个轴是每秒浮点操作数和对主内存访问的浮点操作强度。然而，我们相信Roofline模型可以适用于其他内核，其中性能是不同性能指标的函数。

一个具体的例子是 3-D FFT 的转置阶段，该阶段完全不进行浮点运算。图 5 显示了 Cell 上仅此阶段的 Roofline 模型，其中交换替代了模型中的 Flops。一次交换涉及读取和写入 16 字节，因此其操作强度为 1/32。尽管

computational metric being memory exchanges, note that there is still a computational horizontal Roofline since local stores and caches could affect the number of exchanges that go to DRAM.

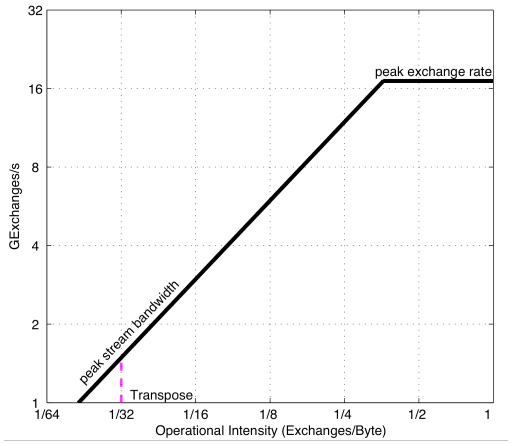

Figure 5. Roofline for transpose phase of 3-D FFT for the Cell

Fallacy: The Roofline model must use DRAM bandwidth.

If the working set fits in the L2 cache, the diagonal Roofline could be L2 cache bandwidth instead of DRAM bandwidth, and the operational intensity on the X-axis would be based on Flops per L2 cache byte accessed. The diagonal memory performance line would move up, and the ridge point would surely move to the left.

For example, Jike Chong ported two financial PDE solvers to four other multicore computers: the Intel Penryn and Larrabee and NVIDIA G80 and GTX280.[9] He used the Roofline model to keep track the platforms' peak arithmetic throughput and L1, L2, and DRAM bandwidths. By analyzing an algorithm's working set and operational intensity, he was able to use the Roofline model to quickly estimate the needs for algorithmic improvements. Specifically, for the option-pricing problem with an implicit PDE solver, the working set is small enough to fit into L1 and the L1 bandwidth is sufficient to support peak arithmetic throughput, so the Roofline model indicates that no optimization is necessary. For option pricing with an explicit PDE formulation, the working set is too large to fit into cache, and the Roofline model helps to indicate the extent to which cache blocking is necessary to extract peak arithmetic performance.

8. CONCLUSIONS

The sea change from sequential computing to parallel computing is increasing the diversity of computers that programmers must confront in making correct, efficient, scalable, and portable software [4]. This paper describes a simple and visual model to help see which systems would be a good match to important kernels, or conversely, to see how to change kernel code or hardware to run desired kernels well. For floating-point kernels that do not fit completely in caches, we showed how *operational intensity*—the number of floating point operations per byte transferred from DRAM—is an important parameter for both the kernels and the multicore computers.

We applied the model to four kernels from the seven dwarfs [10][4] to four recent multicore designs: the AMD Opteron X4, Intel Xeon, IBM Cell, and Sun T2+. The *ridge point*—the minimum operational intensity to achieve maximum performance—proved to be a better predictor of performance than clock rate or peak performance. Cell offered the highest performance on these kernels, but T2+ was the easiest computer on which to achieve its highest performance. One reason is because ridge point of the Roofline model for T2+ was the lowest.

Just the graphical Roofline offers insights into the difficulty of achieving the peak performance of a computer, as it makes obvious when a computer is imbalanced. The operational ridge points for the two x86 computers were 4.4 and 6.7—meaning 35 to 55 Flops per 8-byte operand that accesses DRAM—yet the operational intensities for the 16 combinations of kernels and computers in Table 4 ranged from 0.25 to just 1.64, with a median of 0.60. Architects should keep the ridge point in mind if they want programs to reach peak performance on their new designs.

We measured the roofline and ceilings using microbenchmarks, but we could have used performance counters (see Appendix A.1 and A.3). In fact, we believe there may be a synergistic relationship between performance counters and the Roofline model. The requirements for automatic creation of a Roofline model could guide the designer as to which metrics should be collected when faced with literally hundreds of candidates but a limited hardware budget. [6]

We believe Roofline models can offer insights to other types of multicore systems such as vector processors and GPUs (Graphical Processing Units); other kernels such as sort and ray tracing; other computational metrics such as pair-wise sorts per second and frames per second; and other traffic metrics such as L3 cache bandwidth and I/O bandwidth. Alas, there are many more opportunities than we can pursue. Thus, we invite others to join us in the exploration of the effectiveness of Roofline models.

9. ACKNOWLEDGMENTS

This research was sponsored in part by the Universal Parallel Computing Research Center, funded by Intel and Microsoft, and in part by the ASCR Office in the DOE Office of Science under contract number DE-AC02-05CH11231. We'd like to thank FZ-Jülich and Georgia Tech for access to Cell blades. Our thanks go to Joseph Gebis, Leonid Oliker, John Shalf, Katherine Yelick, and the rest of the Par Lab for feedback on the Roofline model, and to Jike Chong, Kaushik Datta, Mark Hoemmen, Matt Johnson, Jae Lee, Rajesh Nishtala, Heidi Pan, David Wessel, Mark Hill and the anonymous reviewers for feedback on early drafts of this paper.

10. REFERENCES

[1] V. Adve, *Analyzing the Behavior and Performance of Parallel Programs*, PhD thesis, Univ. of Wisconsin, 1993.

[2] AMD, *Software Optimization Guide for AMD Family 10h Processors*, Publication 40546, April 2008.

[3] G. Amdahl, "Validity of the Single Processor Approach to Achieving Large-Scale Computing Capabilities," *AFIPS Conference Proceedings* 30(1967), 483-485.

计算度量为内存交换，请注意仍然存在计算水平Roofline，因为本地存储和缓存可能会影响传输到DRAM的交换数量。

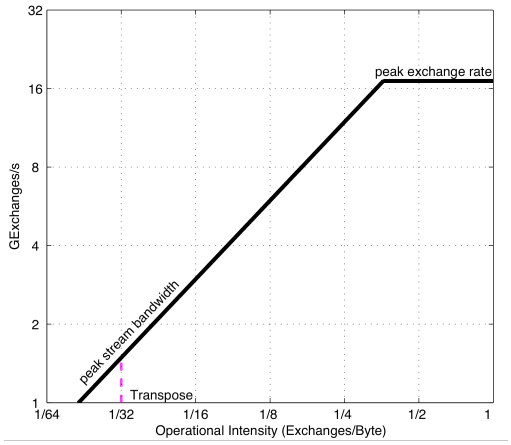

图 5. Cell 的 3-D FFT 转置阶段的 Roofline

Fallacy: The Roofline model must use DRAM bandwidth.

如果工作集适合 L2 缓存，则对角线 Roofline 可能是 L2 缓存带宽，而不是 DRAM 带宽，X 轴上的操作强度将基于每个访问的 L2 缓存字节的 Flops。对角线内存性能线将上移，山脊点肯定会向左移动。

例如，Jike Chong 将两个金融 PDE 求解器移植到四台其他多核计算机：Intel Penryn 和 Larrabee 以及 NVIDIA G80 和 GTX280。[9] 他使用 Roofline 模型来跟踪平台的峰值算术吞吐量以及 L1、L2 和 DRAM 带宽。通过分析算法的工作集和操作强度，他能够利用 Roofline 模型快速估计算法改进的需求。具体来说，对于使用隐式 PDE 求解器的期权定价问题，工作集足够小，可以适应 L1，且 L1 带宽足以支持峰值算术吞吐量，因此 Roofline 模型表明不需要优化。对于使用显式 PDE 公式的期权定价，工作集太大，无法适应缓存，Roofline 模型有助于指示提取峰值算术性能所需的缓存阻塞程度。

8. 结论

从顺序计算到并行计算的海洋变化正在增加程序员在编写正确、高效、可扩展和可移植软件时必须面对的计算机多样性[4]。本文描述了一个简单且直观的模型，以帮助了解哪些系统与重要内核匹配良好，或者反过来，了解如何更改内核代码或硬件以良好运行所需的内核。对于不完全适合缓存的浮点内核，我们展示了 *operational intensity*——从 DRAM 转移的每字节浮点操作数——是内核和多核计算机的一个重要参数。

我们将模型应用于七个小矮人中的四个内核 [10][4]，以及四个最近的多核设计：AMD Opteron X4、Intel Xeon、IBM Cell 和 Sun T2+。*ridge point*——实现最大性能所需的最小操作强度——被证明是比时钟频率或峰值性能更好的性能预测指标。Cell 在这些内核上提供了最高的性能，但 T2+ 是实现其最高性能最容易的计算机。一个原因是 T2+ 的 Roofline 模型的岭点是最低的。

仅仅图形化的 Roofline 提供了关于实现计算机峰值性能难度的见解，因为它清楚地表明了何时计算机处于不平衡状态。两台 x86 计算机的操作岭点分别为 4.4 和 6.7——这意味着每个访问 DRAM 的 8 字节操作数可达到 35 到 55 Flops——然而，表 4 中 16 种内核和计算机组合的操作强度范围从 0.25 到仅 1.64，中位数为 0.60。如果架构师希望程序在他们的新设计上达到峰值性能，则应牢记岭点。

我们使用微基准测试测量了屋顶线和上限，但我们本可以使用性能计数器（见附录 A.1 和 A.3）。事实上，我们相信性能计数器与屋顶线模型之间可能存在协同关系。自动创建屋顶线模型的要求可以指导设计师在面对数百个候选者但硬件预算有限时，应该收集哪些指标。[6]

我们相信，Roofline 模型可以为其他类型的多核系统提供洞察，例如向量处理器和 GPU（图形处理单元）；其他内核，如排序和光线追踪；其他计算指标，如每秒成对排序和每秒帧数；以及其他流量指标，如 L3 缓存带宽和 I/O 带宽。可惜的是，机会远比我们能追求的要多。因此，我们邀请其他人加入我们，探索 Roofline 模型的有效性。

9. 致谢

这项研究部分由英特尔和微软资助的通用并行计算研究中心赞助，部分由美国能源部科学办公室的ASCR办公室资助，合同编号为DE-AC02-05CH11231。我们感谢FZ-Jülich和乔治亚州理工学院提供的Cell刀片。感谢Joseph Gebis、Leonid Oliker、John Shalf、Katherine Yelick以及Par Lab的其他成员对Roofline模型的反馈，以及Jike Chong、Kaushik Datta、Mark Hoemmen、Matt Johnson、Jae Lee、Rajesh Nishtala、Heidi Pan、David Wessel、Mark Hill和匿名评审对本文早期草稿的反馈。

10. 参考文献

[1] V. Adve, *Analyzing the Behavior and Performance of Parallel Programs*，博士论文，威斯康星大学，1993年。[2] AMD，*Software Optimization Guide for AMD Family 10h Processors*，出版物40546，2008年4月。

[3] G. Amdahl，“单处理器方法实现大规模计算能力的有效性，”*AFIPS Conference Proceedings* 30(1967)，483-485。

[4] K. Asanovic, R. Bodik, B. Catanzaro, J. Gebis, K. Keutzer, D. Patterson, W. Plishker, J. Shalf, S. Williams, K. Yelick. “The landscape of parallel computing research: A view from Berkeley.” Tech. Rep. UCB/EECS-2006-183, EECS, U.C. Berkeley, Dec 2006.

[5] C. Bienia,. S. Kumar, J. Singh, and K. Li. “The PARSEC Benchmark Suite: Characterization and Architectural Implications,” Princeton University Technical Report TR-81 1-008, January 2008.

[6] S. Bird et al, “A Case for Sensible Performance Counters,” submitted to the *First USENIX Workshop on Hot Topics in Parallelism (HotPar ’09*), Berkeley CA, March 2009.

[7] E. Boyd, W. Azeem, H. Lee, T. Shih, S. Hung, and E. Davidson, “A Hierarchical Approach to Modeling and Improving the Performance of Scientific Applications on the KSR1,” *Proc. 1994 Int’l Conf. on Parallel Processing*, vol. 3, pp. 188-192, 1994.

[8] D. Callahan, J. Cocke, and K. Kennedy. “Estimating interlock and improving balance for pipelined machines,” *J. Parallel Distrb. Comput.* 5, 334-358. 1988.

[9] J. Chong, Private Communication, 2008.

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

[11] S. Carr and K. Kennedy, “Improving the Ratio of Memory Operations to Floating-Point Operations in Loops,” *ACM TOPLAS* 16(4) (Nov. 1994).

[12] K. Datta, M. Murphy, V. Volkov, S. Williams, J. Carter, L. Oliker, D. Patterson, J Shalf, K. Yelick, “Stencil Computation Optimization and Autotuning on State-of-the-Art Multicore Architectures,” Supercomputing (SC08), 2008.

[13] J. Demmel, J. Dongarra, V. Eijkhout, E. Fuentes, A. Petitet, R. Vuduc, R. Whaley, and K. Yelick, “Self Adapting Linear Algebra Algorithms and Software,” *Proc. IEEE*: *Special Issue on Program Generation, Optimization, and Adaptation*, 93 (2) 2005.

[14] M. Dubois and F. A. Briggs, “Performance of Synchronized Iterative Processes in Multiprocessor Systems,” *IEEE Trans. on Software Engineering* SE-8, 4 (July 1982), 419-431.

[15] M. Frigo and S. Johnson, "The Design and Implementation of FFTW3," *Proc. IEEE: Special Issue on Program Generation, Optimization, and Platform Adaptation.* 93 (2) 2005.

[16] M. Harris, "Mapping Computational Concepts to GPUs," ACM SIGGRAPH Tutorials, Chapter 31, 2005.

[17] J. Hennessy and D. Patterson, *Computer Architecture: A Quantitative Approach*, 4th ed., Boston, MA: Morgan Kaufmann Publishers, 2007.

[18] M. Hill and M. Marty, “Amdahl's Law in the Multicore Era,” *IEEE Computer*, July 2008.

[19] M. Hill and A. Smith, "Evaluating Associativity in CPU Caches," *IEEE Trans. on Computers*, 38(12), pp. 1612-1630, Dec. 1989.

[20] E. Lazowska, J. Zahorjan, S. Graham, and K. Sevcik, *Quantitative System Performance: Computer System Analysis Using Queueing Network Models,* Prentice Hail, Upper Saddle River, NJ, 1984.

[21] J. D. C. Little, "A Proof of the Queueing Formula L = λ W" *Operations Research*, 9, 383-387 (1961).

[22] J. McCalpin, “STREAM: Sustainable Memory Bandwidth in High Performance Computers,” www.cs.virginia.edu/stream, 1995.

[23] D. Patterson, “Latency Lags Bandwidth,” 47:10, *CACM*, Oct. 2004.

[24] S. Williams, *Autotuning Performance on Multicore Computers*, PhD thesis, U.C. Berkeley, 2008.

[25] S. Williams, J. Carter, L. Oliker, J. Shalf, K. Yelick, "Lattice Boltzmann Simulation Optimization on Leading Multicore Platforms," *Int’l Parallel & Distributed Processing Symposium (IPDPS)*, 2008.

[26] S. Williams, L. Oliker, R. Vuduc, J. Shalf, K. Yelick, J. Demmel, "Optimization of Sparse Matrix-Vector Multiplication on Emerging Multicore Platforms," *Supercomputing* (SC07), 2007.

[27] M. Tikir, L. Carrington, E. Strohmaier, A. Snavely, "A Genetic Algorithms Approach to Modeling the Performance of Memory-bound Computations," *Supercomputing* (SC07), 2007.

[28] A. Thomasian and P. Bay, “Analytic Queueing Network Models for Parallel Processing of Task Systems,” *IEEE Trans. on Computers* C-35, 12 (December 1986), 1045-1054.

[29] R. Vuduc, J. Demmel, K. Yelick, S. Kamil, R. Nishtala, and B. Lee. “Performance Optimizations and Bounds for Sparse Matrix-Vector Multiply,” *Supercomputing* (SC02), Nov. 2002.

[30] S. Woo, M. Ohara, E. Torrie, J-P Singh, and A. Gupta. “The SPLASH-2 programs: characterization and methodological considerations,” *Proc. 22nd annual Int’l Symp. on Computer Architecture (ISCA '95),* May 1995, 24 - 36.

Categories and Subject Descriptors

B.8.2 [**Performance And Reliability**]: Performance Analysis and Design Aids, D.1.3 [**Programming Techniques**]: Concurrent Programming

General Terms

Measurement, Performance, Experimentation

Keywords

Performance model, Parallel Computer, Multicore Computer, Multiprocessor, Kernel, Sparse Matrix, Structured Grid, FFT, Stencil, AMD Opteron X4, AMD Barcelona, Intel Xeon, Intel Clovertown, IBM Cell, Sun UltraSPARC T2+, Sun Niagara 2

Appendix A

Appendix A is found online at the CACM website: cacm.acm.org.

[4] K. Asanovic, R. Bodik, B. Catanzaro, J. Gebis, K. Keutzer, D. Patterson, W. Plishker, J. Shalf, S. Williams, K. Yelick. “并行计算研究的全景：来自伯克利的视角。” 技术报告 UCB/EECS-2006-183，EECS，加州大学伯克利分校，2006年12月。

[5] C. Bienia, S. Kumar, J. Singh 和 K. Li. “PARSEC 基准套件：特征化和架构影响，”普林斯顿大学技术报告 TR-81 1-008，2008年1月。

[6] S. Bird 等，“合理性能计数器的案例，” 提交至 *First USENIX Workshop on Hot Topics in Parallelism (HotPar '09)*，加利福尼亚州伯克利，2009年3月。

[7] E. Boyd, W. Azeem, H. Lee, T. Shih, S. Hung, 和 E. Davidson, “一种分层方法来建模和提高 KSR1 上科学应用的性能,” *Proc. 1994 Int'l Conf. on Parallel Processing*, 第 3 卷, 第 188-192 页, 1994.

[8] D. Callahan, J. Cocke, 和 K. Kennedy. “估计互锁并改善流水线机器的平衡，” *J. Parallel Distrb. Comput.* 5, 334-358. 1988.

[9] J. Chong, 私人通信, 2008. [10] P. Colella, “为科学计算定义软件需求,” 演示文稿, 2004. [11] S. Carr 和 K. Kennedy, “提高循环中内存操作与浮点操作的比率,” *ACM TOPLAS* 16(4) (1994年11月). [12] K. Datta, M. Murphy, V. Volkov, S. Williams, J. Carter, L. Oliker, D. Patterson, J Shalf, K. Yelick, “在最先进的多核架构上进行模板计算优化和自调节,” 超级计算 (SC08), 2008. [13] J. Demmel, J. Dongarra, V. Eijkhout, E. Fuentes, A. Petitet, R. Vuduc, R. Whaley, 和 K. Yelick, “自适应线性代数算法和软件,” *Proc. IEEE*: *Special Issue on Program Generation, Optimization, and Adaptation*, 93 (2) 2005. [14] M. Dubois 和 F. A. Briggs, “多处理器系统中同步迭代过程的性能,” *IEEE Trans. on Software Engineering* SE-8, 4 (1982年7月), 419-431. [15] M. Frigo 和 S. Johnson, "FFTW3的设计与实现," *Proc. IEEE: Special Issue on Program Generation, Optimization, and Platform Adaptation.* 93 (2) 2005. [16] M. Harris, "将计算概念映射到GPU," ACM SIGGRAPH教程, 第31章, 2005. [17] J. Hennessy 和 D. Patterson, *Computer Architecture: A Quantitative Approach*, 第4版, 波士顿, MA: Morgan Kaufmann出版社, 2007.

[18] M. Hill 和 M. Marty, “多核时代的阿姆达尔定律，” *IEEE Computer*，2008年7月。

[19] M. Hill 和 A. Smith, "评估 CPU 缓存中的关联性," *IEEE Trans. on Computers*, 38(12), pp. 1612-1630, 1989年12月。

[20] E. Lazowska, J. Zahorjan, S. Graham, 和 K. Sevcik, *Quantitative System Performance: Computer System Analysis Using Queueing Network Models,* Prentice Hail, Upper Saddle River, NJ, 1984. [21] J. D. C. Little, "排队公式 L = λ W" *Operations Research*, 9, 383-387 (1961).

[22] J. McCalpin, “STREAM：高性能中的可持续内存带宽性能计算机，” www.cs.virginia.edu/stream，1995。

[23] D. Patterson, “延迟滞后带宽,” 47:10, *CACM*, 2004年10月。

[24] S. Williams, *Autotuning Performance on Multicore Computers*, 博士论文，加州大学伯克利分校，2008年。

[25] S. Williams, J. Carter, L. Oliker, J. Shalf, K. Yelick, "在领先的多核平台上进行格子玻尔兹曼模拟优化," *Int'l Parallel & Distributed Processing Symposium (IPDPS)*, 2008.

[26] S. Williams, L. Oliker, R. Vuduc, J. Shalf, K. Yelick, J. Demmel, "在新兴多核平台上优化稀疏矩阵-向量乘法," *Supercomputing* (SC07), 2007.

[27] M. Tikir, L. Carrington, E. Strohmaier, A. Snavely, "一种遗传算法方法来建模内存绑定计算的性能," *Supercomputing* (SC07), 2007.

[28] A. Thomasian 和 P. Bay, “任务系统的分析排队网络模型，” *IEEE Trans. on Computers* C- 35, 12 (1986年12月), 1045-1054.

[29] R. Vuduc, J. Demmel, K. Yelick, S. Kamil, R. Nishtala, 和 B. Lee. “稀疏矩阵-向量乘法的性能优化和界限，” *Supercomputing* (SC02)，2002年11月。

[30] S. Woo, M. Ohara, E. Torrie, J-P Singh 和 A. Gupta. “SPLASH-2 程序：特征和方法论考虑，” *Proc. 22nd annual Int'l Symp. on Computer Architecture (ISCA '95),* 1995年5月，24 - 36。

类别和主题描述符

B.8.2 [性能与可靠性]: 性能分析与设计辅助工具, D.1.3 [编程技术]: 并发编程

一般条款

测量，性能，实验

关键词

性能模型，并行计算机，多核计算机，多处理器，内核，稀疏矩阵，结构化网格，快速傅里叶变换，模板，AMD Opteron X4，AMD Barcelona，Intel Xeon，Intel Clovertown，IBM Cell，Sun UltraSPARC T2+，Sun Niagara 2

附录 A

附录 A 可在 CACM 网站上找到：cacm.acm.org。