Impact of Modern Memory Subsystems on Cache Optimizations for Stencil Computations

Shoaib Kamil, Parry Husbands, Leonid Oliker, John Shalf, Katherine Yelick
Lawrence Berkeley National Laboratory
1 Cyclotron Road
Berkeley, CA, 94720
{SAKamil, PJRHusbands, LOliker, JShalf, KAYelick}@lbl.gov *

ABSTRACT

In this work we investigate the impact of evolving memory system features, such as large on-chip caches, automatic prefetch, and the growing distance to main memory on 3D stencil computations. These calculations form the basis for a wide range of scientific applications from simple Jacobi iterations to complex multigrid and block structured adaptive PDE solvers. First we develop a simple benchmark to evaluate the effectiveness of prefetching in cache-based memory systems. Next we present a small parameterized probe and validate its use as a proxy for general stencil computations on three modern microprocessors. We then derive an analytical memory cost model for quantifying cache-blocking behavior and demonstrate its effectiveness in predicting the stencil-computation performance. Overall results demonstrate that recent trends memory system organization have reduced the efficacy of traditional cache-blocking optimizations.

1. INTRODUCTION

Stencil computations on regular grids are at the core of a wide range of scientific applications. In these computations, each point in a multidimensional grid is updated with contributions from a subset of its neighbors. These operations are then used to build solvers that range from simple Jacobi iterations to complex multigrid and block structured adaptive methods.

Reorganizing these computations to take full advantage of memory hierarchies has been the subject of much investigation over the years. These have principally focused on tiling optimizations [1, 2, 3] that attempt to exploit locality by performing operations on cache-sized blocks of data before moving on to the next block. In this work, we re-examine stencil computations on current microprocessors in light of the growing performance gap between processors and memory, as well as the techniques (such as automatic prefetch and large on-chip caches) hardware designers employ to mitigate this problem. Through a combination of techniques, including the use of targeted benchmarks, a parameterized probe, and analytical modeling, we revisit previously successful optimizations and explain their effectiveness (or lack thereof) for three dimensional problems on the current generation of microprocessors.

Our major observation is that improving cache reuse is no longer the dominant factor to consider in optimizing these computations. In particular, streaming memory accesses are increasingly important because they engage software and hardware prefetch mechanisms that are essential to memory performance on modern microprocessors. Many of the grid blocking strategies designed to improve cache locality ultimately end up interfering with prefetch policies and thereby counter the advantages conferred by those optimizations.

The rest of the paper details our methodology, experiments, performance models, and conclusions. In Section 2 we motivate our investigation of modern microprocessor prefetching behavior via the *STriad* microbenchmark. We then derive a simplified analytical model in Section 3 that accurately captures the memory overhead for a given data access pattern. Section 4 presents the Stencil Probe benchmark and validates its use as a proxy for general grid computations on three modern microprocessors. In Section 5, we use the probe to re-examine cache-blocking optimizations that have been employed successfully for stencil computations in previous generations of processor architectures. Next, Section 6 presents a cost model for quantifying cache-blocking behavior and demonstrates its effectiveness in predicting the performance of stencil-based computations. Section 7 discusses the impact of architectural trends in the context of our experimental and analytical results. Finally, we conclude in Section 8 with a summary of our findings and ideas for future work.

2. STANZA TRIAD

In this section we explore prefetching behavior of modern microprocessors using a simple microbenchmark called *Stanza Triad*. An important trend in microprocessor architectures is the attempt to tolerate the increased memory latency relative to clock frequency. Little's Law [4] asserts that in order to fully utilize the total available bandwidth of the memory subsystem, the number of data elements in-flight concurrently must be equal to the product of the bandwidth and the latency of the memory subsys-

*This work was supported in part by the Office of Science, Office of Laboratory Policy and Infrastructure, of the U.S. Department of Energy under contract DE-AC03-76SF00098 through an LBL LDRD grant.

MSP'05, Chicago, USA.

tem. This *bandwidth-delay* product has increased dramatically in recent years. The primary remediation strategy for hiding latency is *prefetch* – both compiler-inserted and automatic hardware prefetch streams. The goal of our work is to demonstrate how the requirements for the efficient use of prefetch can compete with, or even interfere with, traditional strategies employed for cache-blocking, compromising their effectiveness.

To address this issue, we devised a simple microbenchmark called *Stanza Triad*, which is used to evaluate the efficacy of prefetching on various architectures.

2.1 Description

The *Stanza Triad* (STriad) benchmark is a derivative of the STREAM [5] Triad benchmark. STriad works by performing a DAXPY (Triad) inner loop for a size L stanza before jumping k elements and continuing on to the next L elements, until we reach the end of the array. This allows us to then calculate the bandwidth for a particular stanza length L. The minimum stanza length is always greater than or equal to the cache line size of the microprocessor so that cache-line filling efficiency is not a factor in performance. If we set L to the array length, STriad behaves exactly like STREAM Triad.

while $i < arraylength$
    for L elements
        $A_i = scalar * X_i + Y_i$
    skip k elements

Figure 1: STriad pseudocode.

In an architecture with no prefetching, the bandwidth should not be dependent on stanza length (as long as $L > linesize$), since a cache miss will always be incurred at the beginning of each cache line. Loop-overhead of shorter stanzas also will degrade performance, but by comparing our bandwidth numbers to another set of runs with a fixed number of loops irrespective of stanza length, we determined that this effect is relatively minor. In addition, if the bandwidth-delay product is low enough that prefetch provides little benefit, we will see a negligible performance improvement as stanza lengths increase. If, however, the architecture does utilize prefetching technology that effectively masks the bandwidth-delay product, we expect to see improving performance as stanza lengths increase and prefetching engines become engaged.

2.2 Results

Table 1 shows the hardware characteristics of the three prefetch-enabled microprocessors evaluated in out study: Intel Itanium2, AMD Opteron, and IBM PowerPC G5. We also examine an older Intel Pentium3 system that has a bandwidth-delay product that is too small to derive dramatic performance benefits from SSE2 prefetching of streamed data. Notice that the modern processors have significantly larger on-chip caches that operate at full CPU clock rate (3MB, 1MB, 512KB respectively) compared with the Pentium3 system (16KB on-chip, 512KB off-chip operating at half the CPU clock rate). We ran the STriad experiments on the architectures in Table 1, with a total problem size of approximately 48 MB in order to ensure the arrays could not fit in cache. We set k (the jump length) to 2048 which is large enough to ensure no prefetch between stanzas, but small enough to avoid penalties from TLB misses and DDR precharge. Each data size was run multiple times, using a clean cache each time, and we averaged the performance to calculate the memory bandwidth for each stanza length. Results are shown in Figure 2.

	Itanium2	Opteron	PowerPC G5	Pentium 3
clock speed (GHz)	1.3	2.0	2.0	0.45
peak GFLOPS	5.2	4.0	8	1.8
Mem BW (GB/sec)	6.4	6.4	6.4	0.8
STREAM GB/sec	3.85	3.25	2.26	0.26
L1 DCache (KB)	–	64	32	16
L1 Line Size (B)	–	64	128	32
L2 DCache (KB)	256	1024	512	512
L2 Line Size (B)	128	64	128	32
L3 DCache (MB)	3	None	None	None
L3 Line Size (B)	128	None	None	None
Compiler Used	Intel 8.0	PGI 5.2	IBM xlc6/xlf8.1	gcc 3.3.4

Table 1: Summary of Machine Characteristics. L1 information for the Itanium2 is suppressed because the L2 is the closest level of cache that can store FP operands. The Pentium 3 (Katmai) is used for historical comparison; note that the L2 cache on this chip is off-chip and operates at half the core frequency.

On the Opteron and G5 systems, we see a smooth increase in bandwidth until STriad reaches peak. In contrast, the Itanium2 demonstrates a two-phase increase in performance. Although many Itanium2 chipsets do not support hardware prefetch, our test system uses an HP chipset that prefetches in hardware. The slow increase may be due to utilizing only software-initiated prefetches until the number of missed cache lines is sufficient to trigger the hardware prefetch; this then allows the Itanium2 to rapidly increase overall bandwidth until it also reaches peak at $L = 32k$. Unfortunately, facilities (such as performance counters) to directly capture Itanium2 hardware prefetch behavior are not readily available. Lastly, for historical comparison, we ran STriad on a Pentium 3, a system where prefetch does not offer a significant benefit — notice that the performance behavior here is flat and independent of the stanza length. Finally, it can be seen that (as expected) with increasing stanza lengths, STriad performance asymptotically approaches the measured STREAM Triad results.

3. MEMORY MODEL FOR STRIAD

Based on the measured STriad performance, we now formulate a simple model to predict memory access overhead for a given stanza length. We approximate the cost of accessing the first (non-streamed) cache line from main memory, C_{first}, by the overhead of performing an STriad with a short (single cache line) stanza length. C_{stream}, on the other hand, represents the cost of a unit-stride (streamed) cache miss, as computed by performing an STriad where the stanza length is maximized (set to the total array length). The cost of streaming through L words of data, for a given architecture containing W words per cache line, is calculated as $C_{first} + (\lceil (L/W) \rceil - 1) * C_{stream}$. In other words, we assume that after paying C_{first} to bring in the first cache line from main memory, the remaining data accesses cost C_{stream} due to enabled stream prefetching. Note that this simplified approach does not distinguish between the cost of

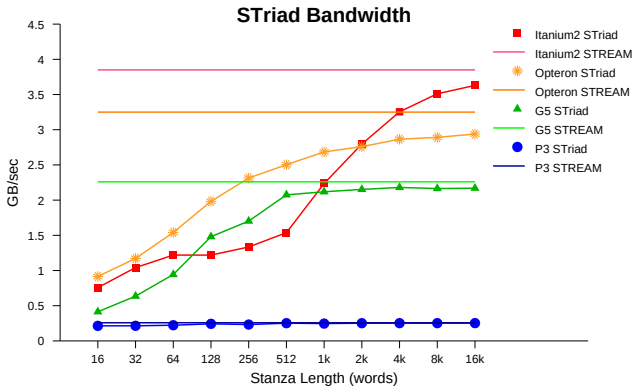

Figure 2: Performance of STriad on the four evaluated architectures. Each achieves within 10% of STREAM Triad peak performance at $L = 32k$. Note that each word is a 64-bit double.

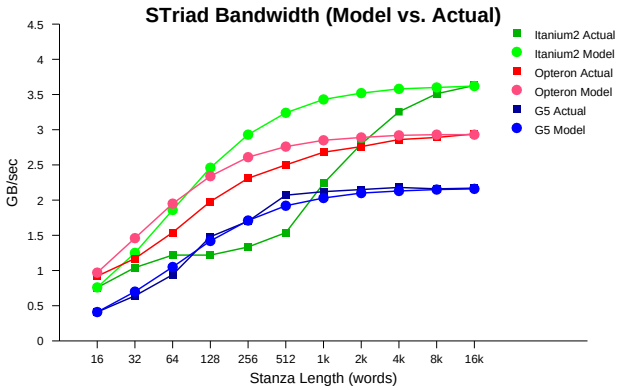

Figure 3: Performance of STriad versus our analytical model on the Itanium2, Opteron, and G5.

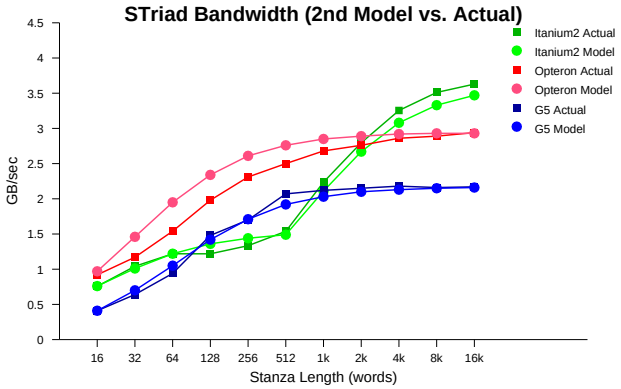

Figure 4: Performance of STriad verus our three-point piecewise linear model on the Itanium2, Opteron, and G5.

loads and stores.

Figure 3 compares the performance between the measured STriad experiments and our analytical model for the three modern microprocessors evaluated in our study. Results show that our simple performance model reasonably approximates the memory access behavior of the G5 and Opteron. However, a large discrepancy is seen for intermediate data sizes on the Itanium2. This is due to the slow increase in performance for stanza lengths between 128 and 2048 words (possibly resulting from the combination of hardware and software prefetching), which is not adequately captured in our two-parameter cost model.

The Itanium2 shows two different performance regimes for prefetch, rather than one, that likely reflect the two different prefetching mechanisms employed by the system we evaluated. The Hewlett Packard system employs a unique chipset that includes automatic prefetch capabilities on the system board that assist the Itanium2 processor's software prefetching capabilities. In order to account for this effect, we use a more sophisticated function to model STriad behavior on the Itanium2. Figure 4 shows that by using a three-point piecewise linear extrapolation, we can approximate STriad behavior with much higher accuracy — especially for the Itanium2 system. However, the simple two parameter model is sufficient for the experiments reported in this paper.

4. STENCIL PROBE

As seen in Section 2, prefetching engines can dramatically affect memory access performance on modern microprocessors. We now wish to apply these lessons to stencil-based numerical computations. However, modifying full-scale applications in order to evaluate various optimization strategies is an extremely time consuming endeavour; we therefore develop a lightweight, flexible stencil-application benchmark called the *Stencil Probe*.

This section first presents a high-level overview of stencil computations, which are an important component of many numerical algorithms. We then introduce the *Stencil Probe*, a parameterized benchmark that mimics the performance of stencil-based computations, and validate its effectiveness at emulating the performance of two important numerical applications. We will then use the Stencil Probe to explore cache blocking optimizations for stencil computations in the next section.

4.1 Stencil Computations

Stencil computations on regular grids are at the core of a wide range of scientific codes. In these computations each point in a multidimensional grid is updated with contributions from a subset of its neighbors. These "sweeps" (updates of all points in the grid according to the computational rule) are then typically used to build solvers for differential equations.

The stencil applications we study come from the Chombo and Cactus toolkits. Chombo [6] is a set of tools for computing solutions of partial differential equations (PDEs) using finite difference methods on adaptively refined meshes. We use a demo application, `heattut`, that is a simple heat equation solver which doesn't use Chombo's more advanced capabilities.

Cactus [7, 8] is another modular open source framework for computational science. It has been successfully used in many areas of astrophysics, most notably in Numerical Relativity [9] and Relativistic Hydrodynamics [10]. The demo code we use here, called `WaveToyC`, is an application exam-

$$X^t_{i,j,k} = X^{t-1}_{i+1,j,k} + X^{t-1}_{i-1,j,k} + X^{t-1}_{i,j+1,k} + X^{t-1}_{i,j-1,k} + X^{t-1}_{i,j,k+1} + X^{t-1}_{i,j,k-1} - 6X^{t-1}_{i,j,k}/factor^2$$

Figure 5: Inner loop for Chombo `heattut`. The Stencil Probe's inner loop was modified to match this.

ple that solves a 3D hyperbolic PDE by finite differencing. The `WaveToyC` Cactus module (known as a *thorn* in Cactus nomenclature) is written in C, and depends on infrastructure provided by the Cactus framework (known as the *flesh* in Cactus nomenclature).

$$\begin{aligned} X^t_{i,j,k} = & \frac{dt^2}{dx^2} * \left(X^{t-1}_{i+1,j,k} + X^{t-1}_{i-1,j,k}\right) \\ + & \frac{dt^2}{dy^2} * \left(X^{t-1}_{i,j+1,k} + X^{t-1}_{i,j-1,k}\right) \\ + & \frac{dt^2}{dz^2} * \left(X^{t-1}_{i,j,k+1} + X^{t-1}_{i,j,k-1}\right) \\ + & factor * X^{t-1}_{i,j,k} - X^{t-2}_{i,j,k} \end{aligned}$$

Figure 6: Inner loop for Cactus `WaveToyC`. The C version of the Stencil Probe was modified so its inner loop matches this.

4.2 Stencil Probe Overview

The Stencil Probe is a small, self-contained serial microbenchmark that we developed as a tool to explore the behavior of grid-based computations. As such it is suitable for experimentation on architectures in varying states of implementation – from production CPUs to cycle-accurate simulators. By modifying the operations in the inner loop of the benchmark, the Stencil Probe can effectively mimic the kernels of applications that use stencils on regular grids. In this way, we can easily simulate the memory access patterns and performance of large applications, as well as use the Stencil Probe as a testbed for potential optimizations, without having to port or modify the entire application.

4.3 Stencil Probe Validation

We modified the inner loop of the Stencil Probe to match Chombo `heattut` and Cactus `WaveToyC`. The results for various grid sizes are shown in the Figure 7. Since both applications utilize array padding, we correspondingly pad the array for the Stencil Probe experiments. We used the Performance API (PAPI) [11] for cycle counts on the Itanium2 and Opteron machines, and Apple's CHUD [12] tools for the G5.

On the Opteron, the `heattut` version of the probe does not match Cactus in terms of the absolute number of total cycles. However, we see that the general performance trend is the same for increasing grid sizes. The Chombo version of the Stencil Probe matches actual performance closely on the Opteron machine. Both the Chombo and Cactus versions of the probe mimic measured performance on the G5 reasonably well. Although the absolute numbers differ, the performance as grid size increases follows the same general trend. On the Itanium2, the probe shows the best performance behavior for both both Cactus and Chombo, even in terms of absolute cycle count.

Overall, these results show that the Stencil Probe is a lightweight tool that can effectively match the performance behavior for varying types of stencil methods and architectural platforms. This allows us to use the Stencil Probe as a proxy for testing architectural features and optimizations for stencil-based applications. We explore one possible optimization, cache blocking, in the next section.

5. CACHE BLOCKING STENCIL CODES

There has been considerable work in memory optimizations for stencil computations, motivated by both the importance of these algorithmic kernels and their poor performance when compared to machine peak. Cache blocking is the standard technique for improving cache reuse, because it reduces the memory bandwidth requirements of an algorithm. Unfortunately, there is limited potential for cache blocking in a single stencil sweep, because each grid value is used a small, constant number of times, which is simply the number of points in the stencil operator.

In an attempt to improve the potential for reuse, several researchers have looked at the idea of merging together multiple sweeps over a regular or irregular mesh [1, 3, 13, 14]. Such techniques have proven effective, although there are many application domains in which they are not applicable. The underlying assumption is that no other computation needs to be performed between consecutive sweeps, whether those sweeps are performing a timestep update, as in an explicit method, or are part of an iterative solver, within a single timestep. There are several reasons why this assumption may not be valid: boundary values may need to be updated based on some non-stencil computation; computation may proceed on a different grid level in a multigrid code; or other physical phenomenon may need to be simulated between timesteps. There are scenarios, such as the application of multiple smoother steps in a multigrid code, where sweeps can be combined, but it is by no means the norm.

Our work considers the more challenging problem of improving memory performance within a single sweep. While the potential payoff for a given optimization is lower, the techniques are more broadly applicable. In prior work, Rivera and Tseng [2] conclude that blocking of 2D applications is not likely to be effective in practice. Our analysis in Section 7 agrees with and further quantifies this result, showing that enormous grids are necessary for 2D cache blocking to be effective on current machines.

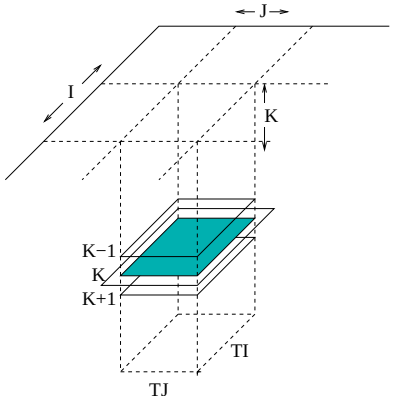

Figure 8: Partial 3-D blocking using a series of 2D slices stacked up in the unblocked dimension, K. Here I is the unit-stride dimension.

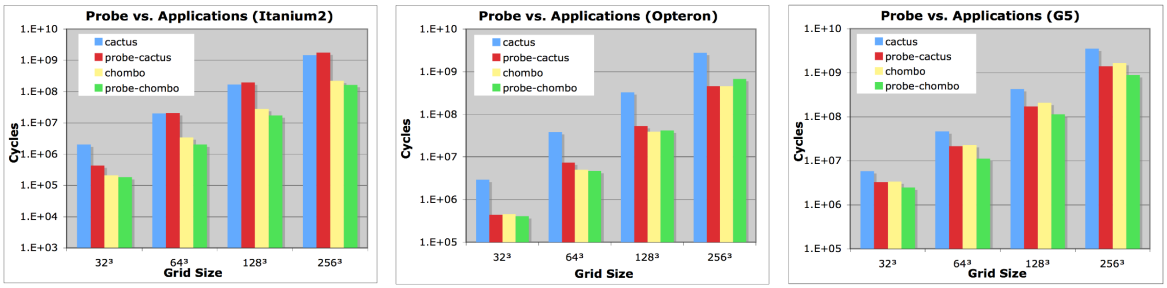

Figure 7: Stencil Probe validation for Chombo and Cactus. The trend of performance emulation with increasing grid sizes shows that the Stencil Probe is a suitable proxy for stencil-type applications.

Rivera and Tseng [2] proposed a blocking scheme for 3D stencil problems that attempts to alleviate the tiny block sizes that result from traditional 2D blocking schemes when applied to three dimensions. Subdividing a 3D grid into cache blocks results in many small blocks because $blocksize^3$ doubles must fit in the cache, as opposed to $blocksize^2$ doubles when blocking in 2D. These small blocking factors cause poor spatial locality because array elements are often accessed in a non-contiguous fashion. Rivera and Tseng attempted to sidestep this limitation by blocking in the two least significant dimensions only (partial 3D blocking). This results in a series of 2D slices that are stacked up in the unblocked dimension, as shown in Figure 8.

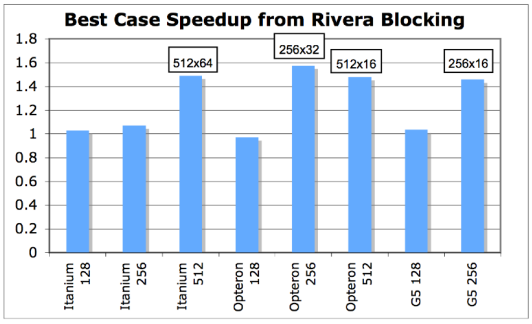

Figure 9: Summary of results of partial 3D blocking for various grid sizes. An exhaustive set of block sizes were examined for each architecture with the best results shown relative to the unblocked version. The optimal block size is shown in the text box.

In order to test the effectiveness of partial 3D blocking, we ran problem sizes up to the largest that would fit in the physical memory of our machines. In Figure 9 we see the best-case cache-blocked results relative to the unblocked version for grid sizes of 128^3, 256^3, and, where applicable, 512^3. The partial 3D blocking speeds up our stencil computation for grid sizes larger than 256^3 on the G5 and the Opteron, while on the Itanium2, speed ups are only seen for a grid size of 512^3. Observe that in all cases where blocking confers an advantage, the I^{th} blocking dimension is equal to the grid size (i.e. maximized).

To further validate the Stencil Probe as well as our experimental results showing speedups at large grid sizes, we modified Cactus `WaveToy` and Chombo `heattut` to use partial 3D blocking. In Figure 11 we see the results of our modification. We acheived a minimum speedup of 10%, while

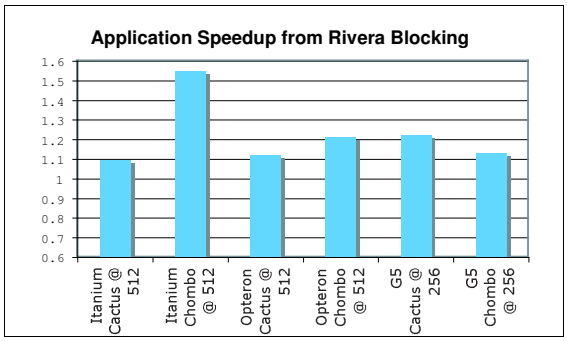

Figure 11: Summary of application performance with partial 3D blocking. In all cases, the application benefitted from the optimization, yielding a minimum of a 10% speedup. The average speedup was 22%.

the average was 22%. We do not expect our applications to reproduce the percentage speedup of the Stencil Probe because applications have much more overhead; however, the fact that each application produced a non-negligible speedup with partial 3D blocking further validates both the Rivera blocking strategy as well as the ability of the Stencil Probe to function as a testbed for stencil code modifications.

In order to understand which blocking factors are the most effective for a given architectural configuration, we construct a simple analytical model to predict the cost of memory traffic for a stencil-based computation.

6. COST MODEL FOR CACHE BLOCKING

We now build on the prefetch-based memory model developed in Section 3 to capture the behavior of stencil computations using various cache-blocking arrangements, as seen in Section 5.

Given an N^3 grid we first approximate the lower and upper bounds on traffic between cache and main memory for a given $I \times J \times N$ blocking. Recall that a 3D stencil calculation accesses 6 columns in three adjacent planes. The lower bound for traffic assumes perfect reuse, where all three $I \times J$-sized planes fit in cache — thus the grid is only transfered twice from main memory (one read and one write). The upper bound (pessimistically) assumes no cache reuse of the $I \times J$ planes due to conflict and capacity misses; therefore, for each sweep of a given plane, the 'front' and 'back' planes required for the 7-point stencil calculation must be reloaded from main memory. The upper bound thus requires the grid

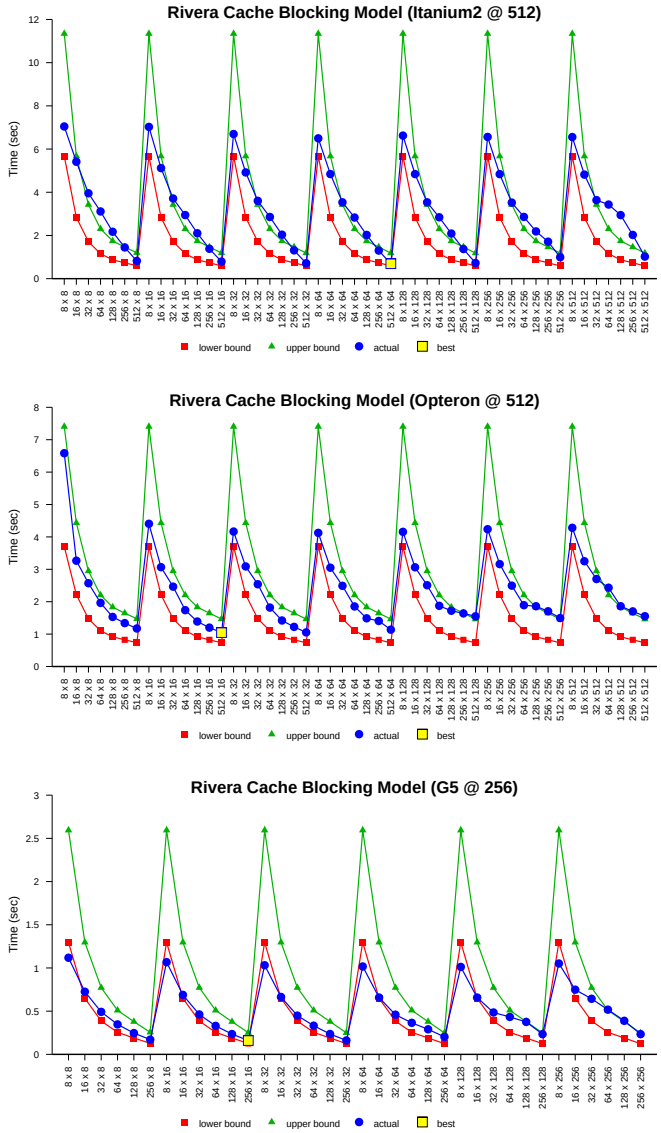

Figure 10: Comparison between partial 3D-blocking runs and the lower/upper bounds of our memory model. Results show that our analytical approach is extremely effective in predicting blocked stencil performance.

to be transferred four times from main memory (three reads and one write): twice the cost of the lower bound. Note that this is not a strict upper bound, since we assume an optimal number of loads and only consider the costs of memory operations (ignoring registers, ALU, etc). Additionally, our simplified performance model does not differentiate between the cost of load and store operations.

Having established lower (2X) and upper (4X) bounds for the grid memory traffic, we now compute the cost of transferring the appropriate stencil data for a given $I \times J \times N$ blocking. Given a system with W words per cache line, a sweep through an N^3 grid requires a total of $T_{total} = \frac{\lceil (I/W) \rceil N^3}{I}$ cache lines. Because the grid is accessed in a blocked fashion, we compute the number of non-streamed (non-prefetched) cache line accesses: $T_{first} = \frac{N^3}{I}$ if $I \neq N$, or $\frac{N^3}{IJ}$ if $I = N \neq J$, or $\frac{N^2}{IJ}$ if $I = J = N$. The total number of streamed (prefetched) cache lines is then the difference between these two values: $T_{stream} = T_{total} - T_{first}$.

We now apply the cost model derived in Section 3, where we established that non-streamed access to a cache line from main memory requires a higher overhead (C_{first}) than subsequently streamed cache lines (C_{stream}), due to the benefits of prefetching. Thus the total cost of sweeping through a 3D stencil in blocked fashion is approximated as $C_{stencil} = C_{first}T_{first} + C_{stream}T_{stream}$. The lower bound of the memory cost for the stencil computation is therefore $2C_{stencil}$, while is upper bound is $4C_{stencil}$. Therefore, setting the block size too small will incur a penalty on memory system performance because prefetch is not engaged.

Figure 10 shows the lower and upper bounds of our cost model compared with the measured results of the Stencil Probe using Rivera blocking across a complete set (powers of two) of $I \times J \times N$ blocking factors. Results show that our

analytical model performs extremely well in capturing the behavior of the stencil computation for all three evaluated architectures. The actual data does occasionally fall outside of the computed lower/upper bounds, but it is clear from the overall performance trends that our methodology is effective in quantifying the tradeoffs between cache blocking and prefetching efficacy.

In the next section, we discuss trends in modern architectures in light of our analysis of partial blocking and the impact of prefetch.

7. IMPACT OF ARCHITECTURAL TRENDS

It is important to understand our cache-blocking findings in the context of evolving architectural trends. As silicon lithography techniques improve, processor architects are able to migrate more levels of the cache hierarchy onto the same chip as the microprocessor core. In addition to reducing the latencies for cache misses at each level of the hierarchy, this has also enabled the designers to operate on-chip caches at the same clock frequency as the core. In these cases, an on-chip L2 (and in the case of the Itanium, the on-chip L3) can deliver operands to the core at the same rate as the L1 caches.

Consider that the 360MHz Sun UltraSparc2i platform, studied in the cache tiling work of Rivera and Tseng [2] (described in Section 5), used a 16KB on-chip L1, but the off-chip L2 operated at half the processor's cycle time. Likewise, the Pentium II that was used to demonstrate another effective blocking technique [1] operated the off-chip L2 at half the clock rate of the on-chip L1. In contrast, all three of the processors reviewed in this paper employ a cache hierarchy that is entirely on-chip and operates at the same clock frequency as the core — allowing each level of the hierarchy to operate at the nearly the same effective bandwidth. Since the on-chip cache sizes (operating at the same clock rate as the core) have increased dramatically in recent processor generations, block-sizes that improve bandwidth locality have increased correspondingly.

The benchmark data in the previous sections suggests that code optimizations should focus on creating the longest possible stanzas of contiguous memory accesses in order to maintain peak performance. These requirements are driven by prefetch behavior, which are fully engaged via long stanzas of unit-stride stream accesses. Thus, in practical terms, stencil computations must be blocked for the largest level of cache hierarchy that operates at core bandwidth, as was empirically and analytically demonstrated in Sections 5 and 6.

Figure 12 describes the conditions where tiling may offer a benefit for 2D and 3D stencil computations, based on our analysis. Six microprocessor architectures are plotted on this graph, based on the the largest tile size that would derive a performance gain for stencil computations. This is equivalent to the deepest level of cache capable of communicating with the processor at full bandwidth. Two different generations of microprocessors are plotted, where the vertical position is based on the on-chip cache size, while the horizontal position is based on the memory footprint for a given sized 3D stencil (128^3, 256^3, and 512^3). Any processors that are below the top red line (bottom blue line) may see a performance benefit from tiling for the 3D (2D) problems. (Note that there is more opportunity for effective cache-blocking for 3D computations than for 2D.) Processors above these lines will likely see a performance degradation from attempts to use a tile-based optimization strategy, since all the appropriate data-sets already fit in the on-chip caches without blocking. It can be seen clearly from this graph that the growth of on-chip L2 and L3 caches have dramatically raised the size of problems that would see any benefit from cache-blocking for stencil computations.

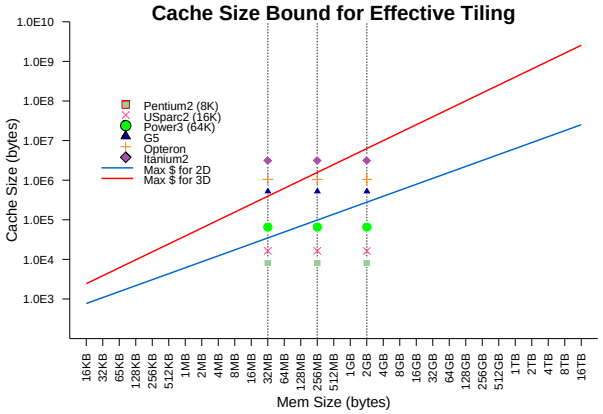

Figure 12: Conditions where tiling offers a potential benefit for 2D and 3D stencil computations. The upper red line (lower blue line) shows the cache limit for a given problem size that could benefit from 3D (2D) blocking. Six microprocessor on-chip cache sizes are plotted. Processors below the line may benefit from cache blocking for the specified problem sizes (128^3, 256^3, 512^3) whereas those above a given line will generally not.

The predictions of Figure 12 can be compared against the results presented Section 5. As Figure 9 shows, the G5 and Opteron clearly benefit from blocking for 256^3 and 512^3 grid sizes, while the Itanium only benefits from blocking for the 512^3 problem size; however, no performance gain is seen for any of the architectures for the 128^3 problem size. This is precisely the behavior predicted in Figure 12!

It is also important to understand the range of grid sizes currently being utilized in large-scale scientific applications. The grid sizes for Cactus are typically 80^3 per processor for parallel General Relativity simulations, and will occasionally be stretched to 128^3 per processor, if the memory is available. Chombo is an AMR code that uses adaptive hierarchical meshes to refine the computation where needed. Cells are selected for refinement on a per-cell basis, which are then aggregated in order to create the largest possible grid that can achieve a specified filling ratio of selected cells. While larger grids can be formed, the typical grid size formed by this strategy is 32^3 to 64^3 elements in size. Thus, it is our observation that high-end stencil computations are currently not run at a grid scale that would benefit from tiling, due to the large on-chip caches of the underlying microprocessors.

8. CONCLUSIONS & FUTURE WORK

In this paper, we investigated the impact of trends in memory subsystems on cache optimizations for stencil computations. Our work is the first to consider the behavior of traditional blocking strategies on prefetch-enabled micro-

processors. We offer several important contributions:

- Modern processors contain on-chip caches of sizes relatively large in comparison to main memory size. This means that cache blocking is now effective only for very large (sometimes unrealistic) problem sizes.
- Prefetching, both in hardware and software, has resulted in improved performance of long stride-1 accesses. This has the effect of making performance more sensitive to discontinuities in the access patterns.
- A simple benchmark, STriad, that tests the effectiveness of prefetch mechanism to hide memory latency.
- The introduction of a parameterized Stencil Probe, that we verify captures the performance of real scientific kernels, which we utilize to test cache blocking optimization strategies on three advanced microprocessors.
- Application of cache-blocking insights from the Stencil Probe to two different applications, yielding significant speedups.
- Derivation of simple analytical memory models for both STriad and for partial 3D blocking on stencil grids. The latter demonstrates the utility of prefetch, and the importance of avoiding cache-blocking in the unit stride direction.

In the future, we will extend our tests to other architectures such as the Power5, which includes more sophisticated software hints for controlling prefetching engines, as well as vector machines that hide latency using deeply pipelined vector register load/store requests. We also plan to utilize the Stencil Probe for application tuning and the evaluation of additional optimizations. In addition, we will further refine our analysis by using detailed information from simulators, allowing us to explore how changes in architectural parameters affect stencil computations. Our long term goal is to develop advanced prefetching methodologies that will address the deficiencies of short stanza accesses as described in this work.

9. ACKNOWLEDGMENTS

The authors would like to thank Paul Hargrove at LBNL and Kaushik Datta at UC Berkeley for their many contributions.

10. REFERENCES

[1] S. Sellappa and S. Chatterjee, "Cache-efficient multigrid algorithms," *International Journal of High Performance Computing Applications*, vol. 18, no. 1, pp. 115–133, 2004.

[2] G. Rivera and C. Tseng, "Tiling optimizations for 3d scientific computations," in *Proceedings of SC'00*, (Dallas, TX), Supercomputing 2000, November 2000.

[3] A. Lim, S. Liao, and M. Lam, "Blocking and array contraction across arbitrarily nested loops using affine partitioning," in *Proceedings of the ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, June 2001.

[4] D. Bailey, "Littleś law and high performance computing," *RNR Technical Report*, 1997.

[5] J. McCalpin, "Memory bandwidth and machine balance in current high performance computers," *IEEE TCAA Newsletter*, December 1995.

[6] "Chombo homepage." `http://seesar.lbl.gov/anag/chombo/`, 2004.

[7] "Cactus Homepage." `http://www.cactuscode.org`, 2004.

[8] W. Benger, I. Foster, J. Novotny, E. Seidel, J. Shalf, W. Smith, and P. Walker, "Numerical relativity in a distributed environment," in *Proceedings of the Ninth SIAM Conference on Parallel Processing for Scientific Computing*, 1999.

[9] M. Alcubierre, G. Allen, B. Brgmann, E. Seidel, and W.-M. Suen, "Towards an understanding of the stability properties of the 3+1 evolution equations in general relativity," *Phys. Rev. D*, vol. (gr-qc/9908079), 2000.

[10] J. A. Font, M. Miller, W. M. Suen, and M. Tobias, "Three dimensional numerical general relativistic hydrodynamics: Formulations, methods, and code tests," *Phys. Rev. D*, vol. Phys.Rev. D61, 2000.

[11] "Performance API homepage." `http://icl.cs.utk.edu/papi`, 2005.

[12] "CHUD homepage." `http://developer.apple.com/tools/performance/`, 2005.

[13] Z. Li and Y. Song, "Automatic tiling of iterative stencil loops," *ACM Trans. Program. Lang. Syst.*, vol. 26, no. 6, pp. 975–1028, 2004.

[14] M. M. Strout, L. Carter, J. Ferrante, J. Freeman, and B. Kreaseck, "Combining performance aspects of irregular gauss-seidel via sparse tiling," in *15th Workshop on Languages and Compilers for Parallel Computing (LCPC)*, (College Park, Maryland), July 25-27, 2002.