LA-UR- 99-1119

Approved for public release; distribution is unlimited.

Title: Optimizing Transformations of Stencil Operations for Parallel Cache-based Architectures

Author(s): Federico Bassetti, CIC-19
Kei Davis, CIC-19

Submitted to: 1999 International Conference on Parallel and Distributed Processing Techniques and Applications
June 28 - July 1, 1999
Monte Carlo Resort, Las Vegas, NV

Los Alamos
NATIONAL LABORATORY

Optimizing Transformations of Stencil Operations for Parallel Cache-based Architectures
ABSTRACT

Federico Bassetti and Kei Davis
Scientific Computing Group, CIC-19,
Los Alamos NM, 87545, USA
Tel: (505) 667-7492, Fax: (505) 667-1126
{fede,kei}@lanl.gov

Keywords: parallel stencil operations, optimizing transformation, parallel cache-based architectures.

Abstract

This paper describes a new technique for optimizing serial and parallel stencil- and stencil-like operations for cache-based architectures. This technique takes advantage of the semantic knowledge implicitly in stencil-like computations. The technique is implemented as a source-to-source program transformation; because of its specificity it could not be expected of a conventional compiler.

Empirical results demonstrate a uniform factor of two speedup. The experiments clearly show the benefits of this technique to be a consequence, as intended, of the reduction in cache misses. The test codes are based on a 5-point stencil obtained by the discretization of the Poisson equation and applied to a two-dimensional uniform grid using the Jacobi method as an iterative solver. Results are presented for a 1-D tiling for a single processor, and in parallel using 1-D data partition. For the parallel case both blocking and non-blocking communication are tested. The same scheme of experiments has been performed for the 2-D tiling case. However, for the parallel case the 2-D partitioning is not discussed here, so the parallel case handled for 2-D is 2-D tiling with 1-D data partitioning.

1 Introduction

All general purpose computer systems, from the ubiquitous PC to the largest supercomputers, rely on a multilevel memory hierarchy to achieve a reasonable fraction of the theoretical performance of the CPU core. Excluding the bottom of the hierarchy (disk storage and backing store) this typically comprises a register file, one or more levels of cache, and a nominal main memory; on the largest of such machines, e.g. the DOE ASCI Blue distributed shared memory multiprocessor, main memory is further divided into local—on-box, faster, smaller–and remote–off-box, slower, larger. It is clear to those that create the large-scale scientific (numerically intensive, array-oriented) applications for such large parallel machines that careful data management with respect to inter-processor communication and the memory hierarchy is critical to achieving acceptable or even useful—literally more than a few percent of peak theoretical—performance.

In the context of larg-scale parallel scientific applications, research and experience has shown that the greatest performance gains are realized by directing the most effort toward efficient utilization of the uppermost levels of the memory hierarchy, namely the register file and the L1 (primary) cache, and data distribution/interprocessor communication. Benchmarking has shown that even more than is the case for more mainstream application areas, for numerical codes the bulk of the computation is represented by a very small fraction of the code.

2 Stencil-like Operations

For several families of numerical algorithms the dominant computational kernels are stencil and stencil-like operations, made conspicuous and amenable to parallelization and optimizing transformation by their repetitive and regular access of (typically) very large arrays.

Numerical algorithms used for solving partial differential equations—multigrid methods and those entailed by linear solvers—are rich with stencil-like operations to effect smoothing and relaxation. The distinguishing characteristic of stencil-like operations is the evaluation of a computationally inexpensive expression, for each element of a (possibly multi-dimensional) array, in which the arguments are all of the elements within a given radius of that element.

In this paper Jacobi relaxation is used as a canonical example, using a two-dimensional array and a five-point stencil (diagonal elements are excluded). A single iteration or *sweep* of the stencil computation is of the form

```
for (int i=1; i != I-1; i++)
 for (int j=1; j != J-1; j++)
   A[i][j] = w1*A[i-1][j] + w2*A[i+1][j] +
             w3*A[i][j-1] + w4*A[i][j+1];
```

where A is dimensioned $[0..n_i,0..n_j]$. Generally several sweeps are made, with A and B swapping roles to avoid copying. In a parallel environment the arrays are typically distributed across multiple processors.

3 Efficiency

Naively implemented, stencil operations are severely memory-bound, hence very inefficient, even in the presence of a cache. In essence the problem is that the arrays, being many times the size of the cache, cycle through the cache with minimal (only the overlapping of the stencil arguments) spatial or temporal reuse. Hardware 'prefetching' of cache lines or blocks, and compiler-inserted prefetch instructions on superscalar architectures, is of little help [1].

Optimizing compilers attempt to improve performance by *tiling*—reordering array access such that one or more blocks remain cache resident for several sweeps. Again, in this context improvement is measurable but minimal [1].

Our target machine is the DOE ASCI Blue SGI Origin 2000 distributed shared memory multiprocessor, for which the N MB L1 cache has access time of one clock cycle; the N MB L2 cache 10 clock cycles, and the local main memory 80 clock cycles. Analysis and experiments show that for perfectly uniform sequential single-array access the amortized cost is approximately 6 cycles per word; for simple Jacobi relaxation approximately 10 cycles. This latter is entirely the cost of memory access: on this superscalar architecture the stencil computation is effectively free, the arithmetic is performed in (instruction-level) parallel with memory accesses. In principle average access time could be close to one cycle.

Elsewhere we have shown how processor time can be traded for communication time [2] for net performance gain, and shown simple sequential optimizations (also applicable in the parallel case), to give significant improvement in performance. Others have also proposed various optimizations in this context [5, 3, 4]. This paper describes a new technique which we call *sliding block temporal tiling*, and give performance figures for a number of architectures: a network of workstations, single workstation, and the ASCI Blue Origin 2000.

4 Sliding Block

Following is a sketch of the algorithm for a 2D tiling for a 2D Jacobi 5-point stencil. The terms are defined as follows. *Mixed_Relaxation* performs the relaxation on points that are on the frontier of adjacent blocks. In order to compute correct results the needed values were previously stored. Thus, one of the four points comes from a different array (Figure 1). Each dimension of the block has an associated storage array. *Jacobi_Relaxation* performs the Jacobi 5-point stencil relaxation using only the interior points of a block. *Store* stores the points on the frontier of a block and its adjacent blocks. *Slide* modifies the position of the block so that the points on the frontier of the current block can overlap with points on the frontier of adjacent blocks.

```
INPUT
     Problem Size: X_size, Y_size
     Total Iteration: T_iter
     Block Iteration: B_iter
     Block Size: X_block, Y_block

REPEAT T_iter / B_iter times
  Move along X dimension in X_block increments
    Move along Y dimension in Y_block increments
      REPEAT B_iter times
        Mixed_Relaxation along X_edge
        Mixed_Relaxation along Y_edge
        Jacobi_Relaxation on
          current block (X_block, Y_block)
        Store Y_edge
        Store X_edge
        Update solution on
          current block (X_block, Y_block)
        Slide current block
```

5 Performance

Only a sketch of performance results are given here. Figure 2 shows the reduction in L1 misses; Figure 3 shows that the optimization gives slightly less than a factor of two improvement over compiler tiling (somewhat better than a factor of two is obtained if the compiler does not perform tiling).

6 Content of Full Paper

The full paper will present more detail on the following: motivation, development, and description of the algorithm, and comparison with related work; the memory-centric performance model and its comparison to empirical results; issues related to the data layout and to determining best blocking partitioning, and a description of our program transformation tool ROSE used to automate this and other optimizations. Performance data on various hardware platforms and with various compilers will be presented as graphs and tables.

7 Future Work

There is scope for generalizing the algorithm to N-dimensional tiling with M ($M \leq N$) data partitioning. While the analytical performance predictions agree with the empirical data, it will be revealing to make use of sophisticated architectural simulators to both study potential performance gains on proposed hypothetical architectures as well as validate the model. Also of interest is the performance impact when using different compilers: though not discussed here, register allocation has a significant impact on both overall performance, and optimized relative to unoptimized performance.

The parallel aspect of this technique is a subject of ongoing investigation. Currently only explicit message passing using the MPI library has been evaluated; in future other paradigms such as threads and OpenMP pragmas will be investigated.

References

[1] Federico Bassetti, Kei Davis, Madhav

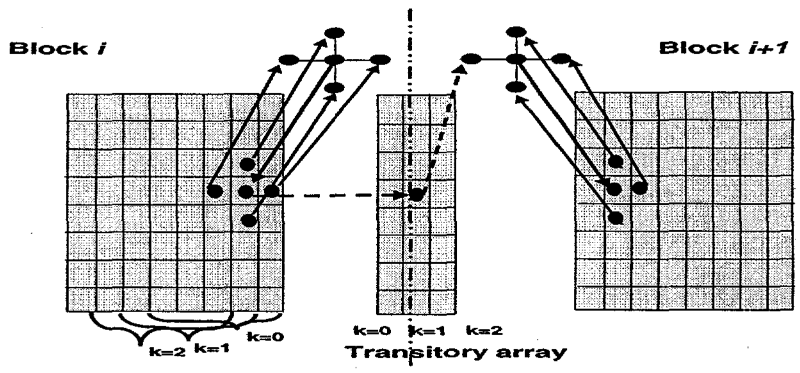

Figure 1: Stencil operation and temporary storage for 1D decomposition of a 2D problem.

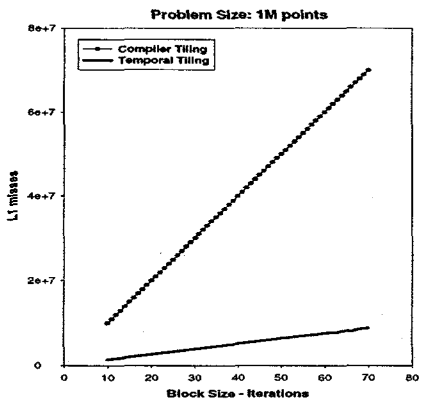

Figure 2: L1 misses as a function of block size/number of iterations.

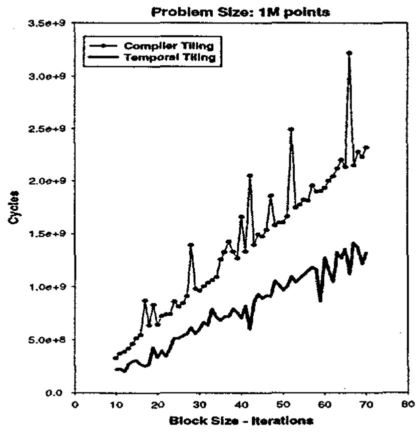

Figure 3: CPU cycles as a function of block size/number of iterations.

Marathe, and Dan Quinlan. Loop transformations for performance and message latency hiding in parallel object-oriented frameworks. In *International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA'98)*, 1998.

[2] Federico Bassetti, Kei Davis, and Dan Quinlan. Optimizing transformations of stencil operations for parallel object-oriented scientific frameworks on cache-based architectures. In D. Caromel et al., editor, *International Scientific Computing in Object-Oriented Parallel Environments, ISCOPE 98*, volume 1505 of *LNCS*. Springer, 1998.

[3] Craig Douglas, Johathan Hu, Ulrich Reude, and Marco Bittencourt. Cache based multigrid on unstructured two dimensional grids, 1998.

[4] Ulrich Reude. Iterative algorithms on high performance architectures. In *EUROPAR'97*, LNCS. Springer Verlag, 1997.

[5] Christian Weiss, Markus Kowarschik, Ulrich Ruede, and Wolfgang Karl. Cache-

aware multigrid methods for solving poisson's equation in two dimensions, 1999.