# Optimizing the NPB MG benchmark for multi-core AMD Opteron microprocessors

Stephen Whalen
Cray, Inc.

June 29, 2007

## 1 Description of MG

### 1.1 High-level description

MG projects the 3D Poisson problem $\nabla^2 u = v$, with periodic boundary conditions, into a trilinear finite element subspace and solves the discrete problem using a V-cycle multigrid algorithm [1]. The Class D problem uses a uniform $1024 \times 1024 \times 1024$ grid to define the hexahedral elements on the finest grid level [7].

### 1.2 Implementational Details

The MPI parallel implementation requires that the number of processes be a power of two. The processes are arranged in a 3D grid, starting with a $1 \times 1 \times 1$-process grid, and successively doubling the number of processes per dimension, first in $z$, then $y$, then $x$, until the grid contains all processes [2].

The trilinear hexahedral finite element space gives rise to a 27-point computational stencil, arranged as a $3 \times 3 \times 3$ cube centered on the current node. This stencil, along with the periodic boundary conditions, requires the augmentation of each process subgrid with two ghost elements per dimension, one in each direction. This results in the local array sizes (for the finest grid level) given in Table 1.

| MPI processes | Process grid | Local array size |
|---|---|---|
| 64 | $4 \times 4 \times 4$ | $258 \times 258 \times 258$ |
| 256 | $4 \times 8 \times 8$ | $258 \times 130 \times 130$ |

Table 1: Per-process array sizes for decomposed MG Class D

## 1.3 Profiles

Portions of function-level sampling profiles are shown in Tables 2 and 3. The `resid` and `psinv` subroutines occupy the top two positions in both profiles, together consuming more than 50% of execution time in both cases. Each routine is dominated by a single triply-nested loop, with near-identical structures in the two subroutines. Therefore, we shall limit our attention to `resid` and `psinv`.

It is clear upon inspection that these loops are ready candidates for cache tiling, as the arrays involved are too large to fit in cache (a $258 \times 130 \times 130$-word array consumes over 33 megabytes), and the data is accessed using the aforementioned 27-point stencil [6].

| Samp % | Cum. Samp % | Samp | Imb. Samp | Imb. Samp % | Function |
|---|---|---|---|---|---|
| 100.0% | 100.0% | 1073683 | -- | -- | Total |
| 45.6% | 45.6% | 489779 | 282.20 | 3.6% | resid_ |
| 20.9% | 66.5% | 224624 | 78.25 | 2.2% | psinv_ |
| 9.3% | 75.9% | 100215 | 17.14 | 1.1% | interp_ |
| 7.6% | 83.5% | 81509 | 42.42 | 3.3% | rprj3_ |
| 5.4% | 88.8% | 57586 | 65.22 | 6.9% | __c_mzero8 |
| 3.0% | 91.8% | 32275 | 138.70 | 21.9% | PtlEQPeek |
| 2.9% | 94.7% | 31057 | 49.73 | 9.4% | take3_ |
| 1.8% | 96.5% | 19575 | 48.14 | 13.8% | give3_ |

Table 2: CrayPat sampling profile for MG Class D, 64 processes

| Samp % | Cum. Samp % | Samp | Imb. Samp | Imb. Samp % | Function |
|---|---|---|---|---|---|
| 100.0% | 100.0% | 1289551 | -- | -- | Total |
| 38.9% | 38.9% | 501084 | 262.64 | 11.9% | resid_ |
| 13.7% | 52.6% | 176756 | 42.55 | 5.8% | psinv_ |
| 12.3% | 64.8% | 158073 | 103.53 | 14.4% | __c_mzero8 |
| 9.7% | 74.5% | 125411 | 132.11 | 21.3% | PtlEQPeek |
| 7.0% | 81.6% | 90796 | 15.33 | 4.2% | interp_ |
| 5.0% | 86.6% | 64998 | 22.10 | 8.0% | rprj3_ |
| 2.4% | 89.0% | 31109 | 54.48 | 31.1% | PtlEQGet |
| 2.3% | 91.3% | 29554 | 31.55 | 21.5% | take3_ |
| 2.1% | 93.4% | 27004 | 27.52 | 20.8% | give3_ |

Table 3: CrayPat sampling profile for MG Class D, 256 processes

# 2 Cache Tiling

As the finest level consumes the majority of the computational resources, our optimization efforts will focus only on the array sizes listed in Table 1.

Rivera and Tseng have completed a rather extensive study of the effects of tiling optimizations on the `resid` kernel, showing that automatic tiling and padding transformations can provide substantial speedups [6]. Our approach is much more intuitive than that of Rivera and Tseng, while still creating substantial gains.

While Rivera and Tseng apply cache-tiling transformations to the two innermost loops, creating blocks extending along the $z$ dimension, we tile the outermost loops [4]. Allowing the blocks to extend along the $x$ dimension preserves the spatial locality inherent in the data layout, as each rank-one slice along the array's $x$ dimension is contiguous in memory.

Thus, for both the 64- and 256-process cases, we are creating blocks extending 258 words in the $x$ dimension. Let $n_y$ and $n_z$ be the blocks' extents in the $y$ and $z$ dimensions, respectively. (The variables are named, respectively, `BLOCK2` and `BLOCK3` in the code fragment shown in Figure 1.) Each block will then contain $258 n_y n_z$ words. The Opteron's 1 MB L2 cache holds 131,072 words. Therefore, to fit a single block within L2, it suffices to take

$$n_y n_z \leq \left\lfloor \frac{131{,}072}{258} \right\rfloor = 508.$$

Giving preference to square tiles over rectangular tiles [3, 5], we select $n_y = 22$ and $n_z = 23$.

The resulting code transformation appears in Figure 1.

| **Reference code** | **Cache-tiling transformation** |
|---|---|
| | `integer i2block, i3block`<br>`integer BLOCK2, BLOCK3`<br>`parameter ( BLOCK2 = 22, BLOCK3 = 23 )`<br>`do i3block=2,n3-1,BLOCK3`<br>`  do i2block=2,n2-1,BLOCK2` |
| `do i3=2,n3-1`<br>`  do i2=2,n2-1`<br>`  do i1=1,n1`<br>*[compute intermediate finite differences]*<br>`  enddo`<br>`  do i1=2,n1-1`<br>*[apply stencil]*<br>`  enddo`<br>`  enddo`<br>`enddo` | `  do i3=i3block,min(i3block+BLOCK3-1,n3-1)`<br>`  do i2=i2block,min(i2block+BLOCK2-1,n2-1)`<br>`  do i1=1,n1`<br>*[compute intermediate finite differences]*<br>`  enddo`<br>`  do i1=2,n1-1`<br>*[apply stencil]*<br>`  enddo`<br>`  enddo`<br>`  enddo`<br>`enddo` |

Figure 1: Cache tiling the outer two loops of `resid` and `psinv`. The inner loops over `i1` are unchanged.

# 3 Results

Performance data were gathered on franklin under Compute Node Linux, with two MPI processes mapped to each node, one process per core. The transformations described in Section 2 provide between 20% and 30% gain in the benchmarks' self-reported Mop/s rates, shown in Table 4.

| **MPI processes** | **Reference code** (Mop/s/process) | **With cache tiling** (Mop/s/process) | **Speedup** |
|---|---|---|---|
| 64 | 754.30 | 976.82 | 29.5% |
| 256 | 675.02 | 821.73 | 21.7% |

Table 4: Performance results before and after cache-tiling transformations

Tables 5 and 6 show hardware counter data for runs using reference and transformed code. In this data, we see that the cache-tiling transformations produce lower L2 and D-TLB miss rates, as expected. This results in large reductions in the time spent stalled waiting for the load-store unit, as well as reducing the time that the FPUs are stalled.

| **Subroutine `resid`** | | |
|---|---|---|
| | Reference code | With cache tiling |
| L1 D-cache accesses | 44724242053 ops | 44745835841 ops |
| L1 D-cache misses that hit in L2 | 1122583735 refills | 1218698984 refills |
| L1 D-cache misses that miss in L2 | 50517790 refills | 34236723 refills |
| D-TLB misses | 16011726 misses | 15150023 misses |
| HW FP Ops | 28135094524 ops | 28135498558 ops |
| HW FP Ops / User time | 893.911 M/sec | 1259.13 M/sec |
| LD & ST per TLB miss | 2793.23 refs/miss | 2953.53 refs/miss |
| User time | 30.481 secs | 22.327 secs |
| Avg Time FPUs stalled | 2.334 secs | 1.765 secs |
| Avg Time LSs stalled | 6.930 secs | 3.576 secs |
| **Subroutine `psinv`** | | |
| | Reference code | With cache tiling |
| L1 D-cache accesses | 21949621061 ops | 21967337840 ops |
| L1 D-cache misses that hit in L2 | 476605356 refills | 518925270 refills |
| L1 D-cache misses that miss in L2 | 26067293 refills | 13499799 refills |
| D-TLB misses | 6049978 misses | 5770389 misses |
| HW FP Ops | 15720443122 ops | 15718609826 ops |
| HW FP Ops / User time | 1099.45 M/sec | 1627.82 M/sec |
| LD & ST per TLB miss | 3628.07 refs/miss | 3806.91 refs/miss |
| User time | 14.299 secs | 9.656 secs |
| Avg Time FPUs stalled | 1.995 secs | 0.780 secs |
| Avg Time LSs stalled | 2.925 secs | 1.622 secs |

Table 5: Hardware counter data for MG Class D, 64 processes, reported as per-process averages.

| **Subroutine `resid`** | | |
|---|---|---|
| | Reference code | With cache tiling |
| L1 D-cache accesses | 11182362630 ops | 11187063000 ops |
| L1 D-cache misses that hit in L2 | 294300679 refills | 312142255 refills |
| L1 D-cache misses that miss in L2 | 12826933 refills | 8914597 refills |
| D-TLB misses | 3481736 misses | 3386057 misses |
| HW FP Ops | 7034187710 ops | 7034220787 ops |
| HW FP Ops / User time | 932.624 M/sec | 1238.65 M/sec |
| LD & ST per TLB miss | 3211.71 refs/miss | 3303.87 refs/miss |
| User time | 7.542 secs | 5.679 secs |
| Avg Time FPUs stalled | 0.509 secs | 0.376 secs |
| Avg Time LSs stalled | 1.933 secs | 0.969 secs |
| **Subroutine `psinv`** | | |
| | Reference code | With cache tiling |
| L1 D-cache accesses | 5493768020 ops | 5499793515 ops |
| L1 D-cache misses that hit in L2 | 128865773 refills | 135603266 refills |
| L1 D-cache misses that miss in L2 | 3119244 refills | 3363692 refills |
| D-TLB misses | 1230175 misses | 1246810 misses |
| HW FP Ops | 3928983156 ops | 3928522452 ops |
| HW FP Ops / User time | 1409.43 M/sec | 1616.51 M/sec |
| LD & ST per TLB miss | 4465.82 refs/miss | 4411.09 refs/miss |
| User time | 2.788 secs | 2.430 secs |
| Avg Time FPUs stalled | 0.316 secs | 0.194 secs |
| Avg Time LSs stalled | 0.470 secs | 0.402 secs |

Table 6: Hardware counter data for MG Class D, 256 processes. Data is reported as the per-process average.

# References


[1] D. Bailey, E. Barszcz, J. Barton, D. Browning, R. Carter, L. Dagum, R. Fatoohi, S. Fineberg, P. Frederickson, T. Lasinski, R. Schreiber, H. Simon, V. Venkatakrishnan, and S. Weeratunga. The NAS parallel benchmarks. Report RNR-94-007, NASA Advanced Supercomputing Division, March 1994.

[2] David Bailey, Tim Harris, William Saphir, Rob van der Wijngaart, Alex Woo, and Maurice Yarrow. The NAS parallel benchmarks 2.0. RNR Technical Report NAS-95-020, NASA Advanced Supercomputing Division, December 1995.

[3] S. Coleman and K. S. McKinley. Tile size selection using cache organization and data layout. In *Proceedings of the SIGPLAN '95 Conference on Programming Language Design and Implementation*, La Jolla, CA, June 1995.

[4] John Levesque, Jeff Larkin, Martyn Foster, Joe Glenski, Gary Geissler, Stephen Whalen, Brian Waldecker, Jonathan Carter, David Skinner, Helen He, Harvey Wasserman, John Shalf, Hongzhang Shan, and Erich Strohmaier. Understanding and mitigating multicore performance issues on the AMD Opteron™ architecture. Technical Report LBNL-62500, Lawrence Berkeley National Laboratory, March 2007.

[5] Gabriel Rivera and Chau-Wen Tseng. A comparison of compiler tiling algorithms. In *Proceedings of the 8th International Conference on Compiler Construction (CC'99)*, Amsterdam, Netherlands, March 1999.

[6] Gabriel Rivera and Chau-Wen Tseng. Tiling optimizations for 3d scientific computations. In *Supercomputing 2000*, Dallas, TX, December 2000.

[7] Rob van der Wijngaart. NAS parallel benchmarks version 2.4. NAS Technical Report NAS-02-007, NASA Advanced Supercomputing Division, October 2002.