Performance Analysis and Optimization of the Trilinos Epetra Package on the Quad-Core AMD Opteron Processor

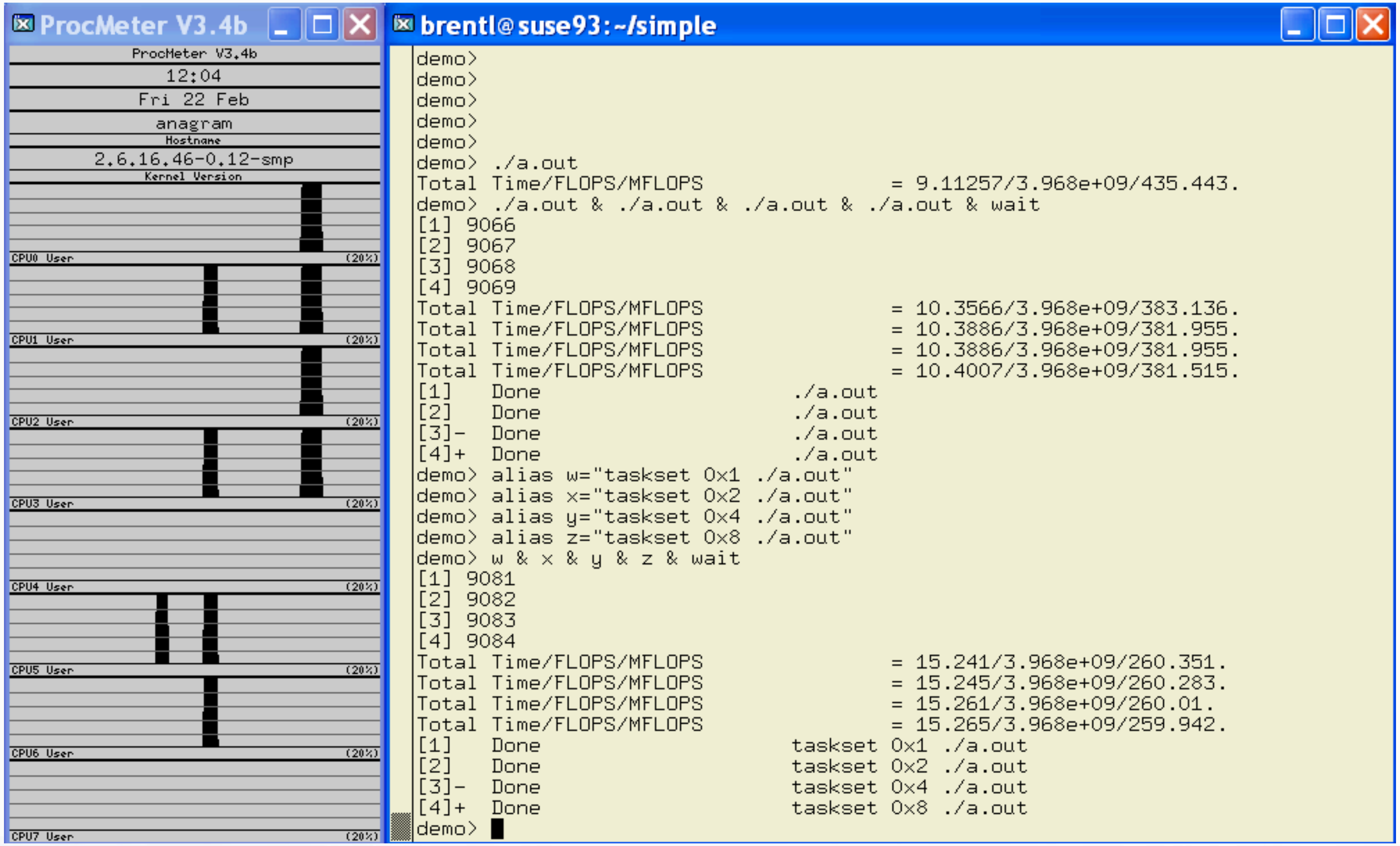
Brent Leback – brent.leback@pgroup.com
Doug Doerfler – dwdoerf@sandia.gov
Michael Heroux – maherou@sandia.gov

CUG Helsinki
May, 2008

The Portland Group

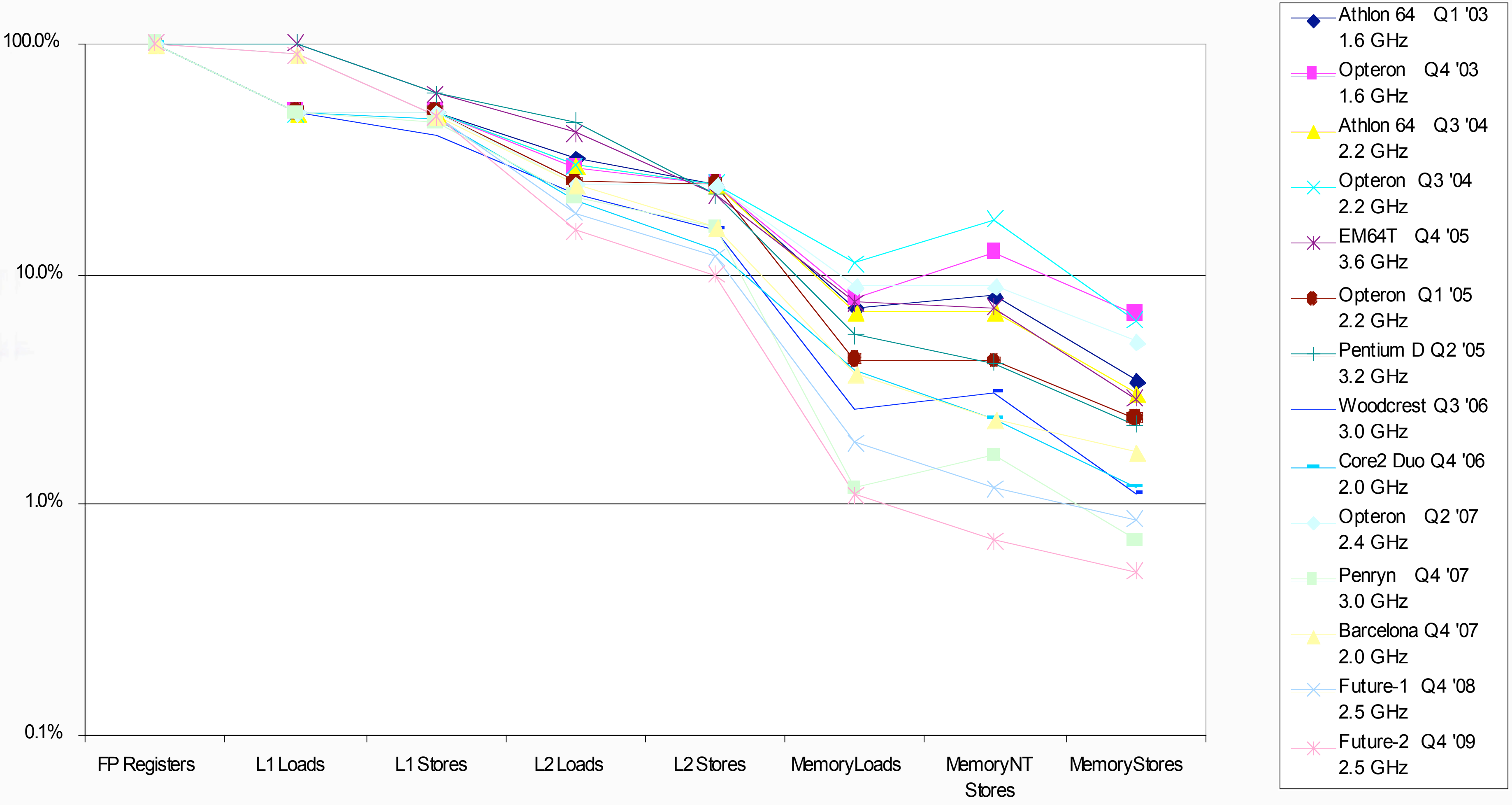
Sandia National Laboratories

Standardizing Quad-Core Performance



```
demo>
demo>
demo>
demo>
demo>
demo> ./a.out
Total Time/FLOPS/MFLOPS               = 9.11257/3.968e+09/435.443.
demo> ./a.out & ./a.out & ./a.out & ./a.out & wait
[1] 9066
[2] 9067
[3] 9068
[4] 9069
Total Time/FLOPS/MFLOPS               = 10.3566/3.968e+09/383.136.
Total Time/FLOPS/MFLOPS               = 10.3886/3.968e+09/381.955.
Total Time/FLOPS/MFLOPS               = 10.3886/3.968e+09/381.955.
Total Time/FLOPS/MFLOPS               = 10.4007/3.968e+09/381.515.
[1]   Done                    ./a.out
[2]   Done                    ./a.out
[3]-  Done                    ./a.out
[4]+  Done                    ./a.out
demo> alias w="taskset 0x1 ./a.out"
demo> alias x="taskset 0x2 ./a.out"
demo> alias y="taskset 0x4 ./a.out"
demo> alias z="taskset 0x8 ./a.out"
demo> w & x & y & z & wait
[1] 9081
[2] 9082
[3] 9083
[4] 9084
Total Time/FLOPS/MFLOPS               = 15.241/3.968e+09/260.351.
Total Time/FLOPS/MFLOPS               = 15.245/3.968e+09/260.283.
Total Time/FLOPS/MFLOPS               = 15.261/3.968e+09/260.01.
Total Time/FLOPS/MFLOPS               = 15.265/3.968e+09/259.942.
[1]   Done                    taskset 0x1 ./a.out
[2]   Done                    taskset 0x2 ./a.out
[3]-  Done                    taskset 0x4 ./a.out
[4]+  Done                    taskset 0x8 ./a.out
demo> █
```

Peak Transfer Rate divided by Peak FP Performance



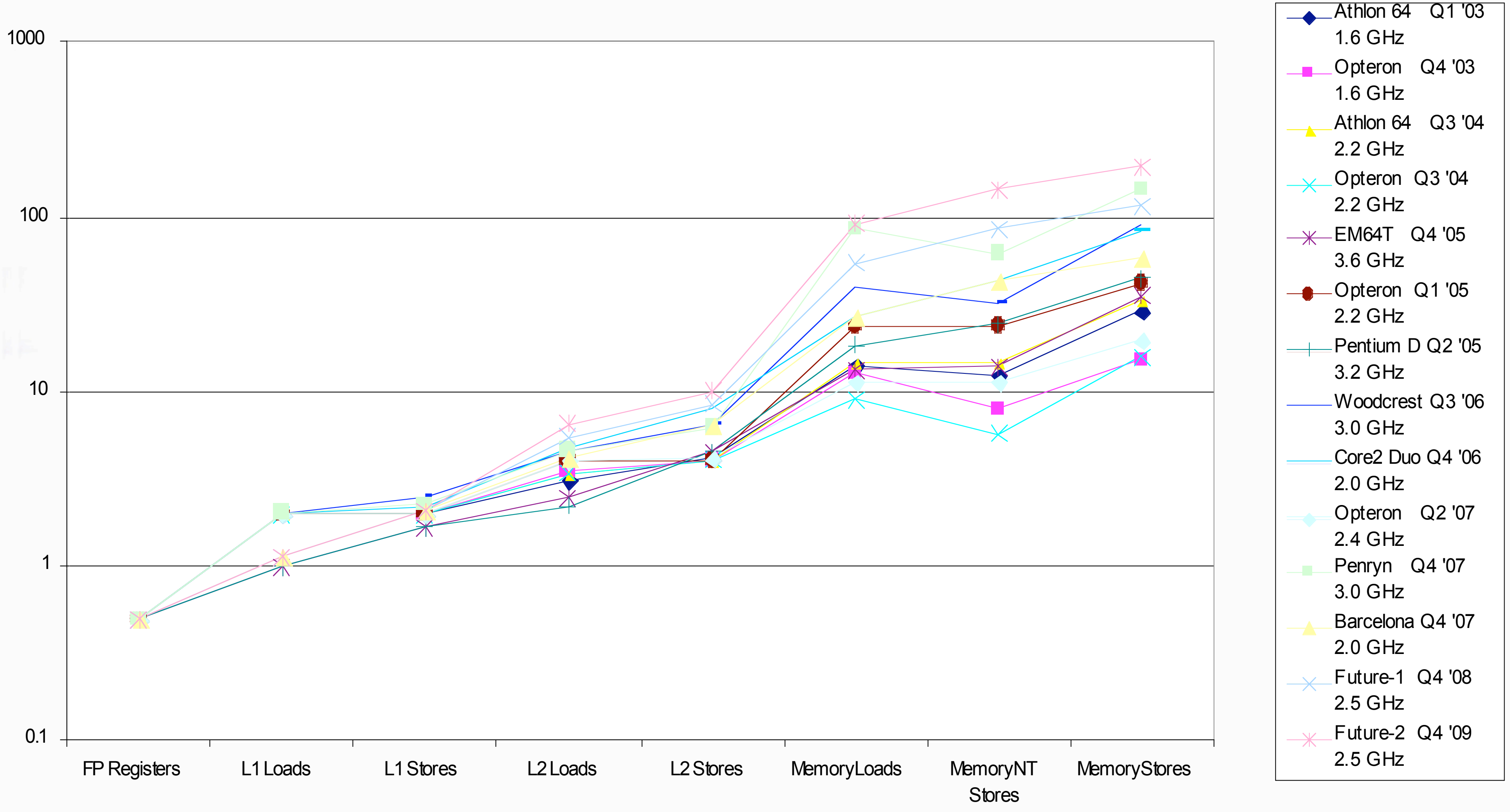
Compute Intensity & Potential Performance

$$\textit{Compute Intensity} = \frac{\text{Total Number of Operations}}{\text{Number of Input/Output Data Points}}$$

$$GFLOPS = Intensity \times Bandwidth$$

$$\frac{GigaFloatingOps}{Second} = \frac{FloatingOps}{\cancel{Word}} \times \frac{Giga\cancel{Words}}{Second}$$

Compute Intensity Required to Sustain Floating-Point Speed



The Portland Group

Sandia National Laboratories

Compute Intensity Given by Compiler

```
dvadd:
     3,  Intensity = 0.333
         double add:1 ld:2 st:1
daxpy:
    11, Intensity = 0.667
         double add:1 mul:1 ld:2 st:1
ddot:
    21, Intensity = 1.000
         double add:1 mul:1 ld:2
```

Targeting Trilinos Epetra Performance

- Epetra is a collection of distributed data objects for sparse and dense matrices, vectors and graphs. It is the most heavily used package in Trilinos because it provides matrix and vector services for all other Trilinos packages.
- The most common sparse matrix operation by far is sparse matrix times a dense vector, sometimes referred to as SpMV, and usually formulated as y = A*x.

Compressed Row Storage Data Format

$$A = \begin{bmatrix} 11 & 12 & 0 & 14 & 15 \\ 21 & 22 & 0 & 0 & 25 \\ 0 & 32 & 33 & 34 & 0 \\ 41 & 0 & 0 & 44 & 0 \\ 0 & 52 & 0 & 0 & 55 \end{bmatrix}$$

$$values = [11,12,14,15 : 21,22,25 : 32,33,34 : 41,44 : 52,55]$$

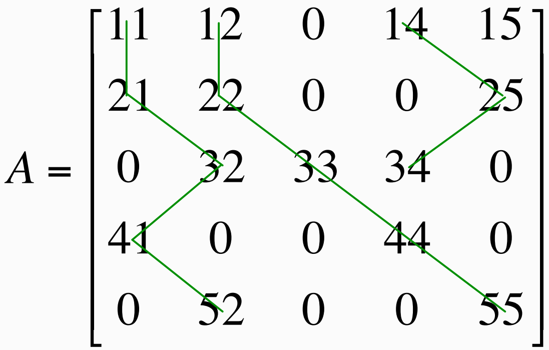
$$indices = [0,1,3,4 : 0,1,4 : 1,2,3 : 0,3 : 1,4]$$

$$offsets = [0,4,7,10,12,14]$$

SpMV:

```
for (i=0; i< nrow; i++) {
  double sum = 0.0;
  double * A_vals = A->ptr_to_vals_in_row[i];
  int    * inds   = A->ptr_to_inds_in_row[i];
  int cur_nnz      = A->nnz_in_row[i];
  for (j=0; j< cur_nnz; j++)
    sum += A_vals[j]*x[inds[j]];
  y[i] = sum;
}
```

Jagged Diagonal Storage Data Format



$$A = \begin{bmatrix} 11 & 12 & 0 & 14 & 15 \\ 21 & 22 & 0 & 0 & 25 \\ 0 & 32 & 33 & 34 & 0 \\ 41 & 0 & 0 & 44 & 0 \\ 0 & 52 & 0 & 0 & 55 \end{bmatrix}$$

$$values = \left[11,21,32,41,52:12,22,33,44,55:14,25,34:15\right]$$

$$indices = \left[0,0,1,0,1:1,1,2,3,4:3,4,3:4\right]$$

$$offsets = \left[0,5,10,13,14\right]$$

```
if (!TransA) {
  for (int i=0; i<jaggedDiagonalLength; i++) {
      int ix = curIndices[i];
      int iy = RowPerm[i];
      double val = curValues[i];
      y[iy] += val*x[ix];
  }
} else {
  for (int i=0; i<jaggedDiagonalLength; i++) {
      int iy = curIndices[i];
      int ix = RowPerm[i];
      double val = curValues[i];
      y[iy] += val*x[ix];
  }
}
```

Sparse Diagonal Storage Data Format

$$A = \begin{bmatrix} 11 & 12 & 0 & 14 & 15 \\ 21 & 22 & 0 & 0 & 25 \\ 0 & 32 & 33 & 34 & 0 \\ 41 & 0 & 0 & 44 & 0 \\ 0 & 52 & 0 & 0 & 55 \end{bmatrix}$$

$$values[0] = [41,52]$$
$$values[1] = [21,32,0,0]$$
$$values[2] = [11,22,33,44,55]$$
$$values[3] = [12,0,34,0]$$
$$values[4] = [14,25]$$
$$values[5] = [15]$$
$$offsets \quad = [-3,-1,0,1,3,4]$$

```
for (int i=0; i<numDiags; i++) {
  curValues = ptr_to_diags[i];
  curDiagOffset = diagonal_offsets[i];
  if (curDiagOffset < 0)
     y = rvector-curDiagOffset;
  else
     y = rvector;
  if (curDiagOffset < 0)
     x = dvector;
  else
     x = dvector+curDiagOffset;
  diagLength = diagonal_lengths[i];
  for (int j=0; j<diagLength; j++) {
     y[j] += curValues[j] * x[j];
  }
}
```

Modifications to Tune this Kernel

```
#define STRIPVAL 16384
 for (int k=0; k<maxDiagLength; k+=STRIPVAL) {
   for (int i=0; i<numDiags; i++) {
     curValues = ptr_to_diags[i];
     curDiagOffset = diagonal_offsets[i];
     y = …;
     x = …;
     diagLength = diagonal_lengths[i];
     curValues += k;
     y += k;
     x += k;
     diagLength -= k;
     if (diagLength > STRIPVAL)
       diagLength = STRIPVAL;
     for (int j=0; j<diagLength; j++) {
#pragma mem prefetch curValues[j+8]
        y[j] += curValues[j] * x[j];
 } } }
```

- **Added restrict qualifers to declarations**
- **Added a strip-mined loop to enable cache reuse on y and x**
- **Added a prefetch pragma so we only prefetch from the A matrix, not y and x**

Results, 1 core per socket, and fully subscribed (4 cores per socket)

1core: Problem Size	24**3	48**3	72**3	100**3
g++ original code, CRS	512	486	466	469
g++ original code, SDS	512	350	267	285
pgCC original code, CRS	487	461	451	430
pgCC original code, SDS	541	360	279	292
pgCC SDS, + restrict	389	343	334	326
g++, SDS, restrict + Strip-mine	540	486	482	493
pgCC SDS, restrict + Strip-mine	389	341	336	346
pgCC same, + -Mnoprefetch	572	459	465	481
pgCC, same + prefetch pragma	695	632	563	607

4 cores: Problem Size	24**3	48**3	72**3	100**3
g++ original code, CRS	314	275	270	271
g++ original code, SDS	442	125	125	128
pgCC original code, CRS	291	271	263	264
pgCC original code, SDS	438	125	126	129
pgCC SDS, + restrict	177	149	135	133
g++, SDS, restrict + Strip-mine	469	353	349	349
pgCC SDS, restrict + Strip-mine	172	148	145	146
pgCC same, + -Mnoprefetch	476	351	351	352
pgCC, same + prefetch pragma	599	422	388	395

Conclusions

- Compiler Feedback: a positive force in HPC SW Evolution
- FLOPS are Free, Bandwidth is Precious
- Design algorithms that minimize data movement and maximize data movement efficiency, rather than minimizing computations
- Use pragmas for fine-tuned control over memory-tuning optimization. One or fewer “streams” per loop is best.
- Strip-mining or other caching techniques are important.