PGI® Compilers and Tools
PGI Premier Support Success Stories

November 2008
ECMWF

Dave Norton – dave.norton@pgroup.com
530.544.9075
www.pgroup.com

The Portland Group

Outline of Today's Topics

- What is PGI Premier Support?
- What is the motivation behind Premier Support
- Alegra kernel work
- Trilinos
- WRF
- Questions and Answers

What is PGI Premier Support?

- PGI Premier Support is a professional services program offered to select customers with the intent of direct engineer to engineer engagement on mission critical customer issues.
- Program components include:
  - PGI Quick Start Seminar with additional customized training options
  - A designated PGI technical contact within engineering
  - PGI Tracker online inquiry tracking system
  - Custom software patches and workarounds
  - Interim PGI releases to address critical issues
  - Custom libraries for runtime debugging
  - Custom application performance analysis and tuning
  - Custom compiler features

PGI Classroom Training

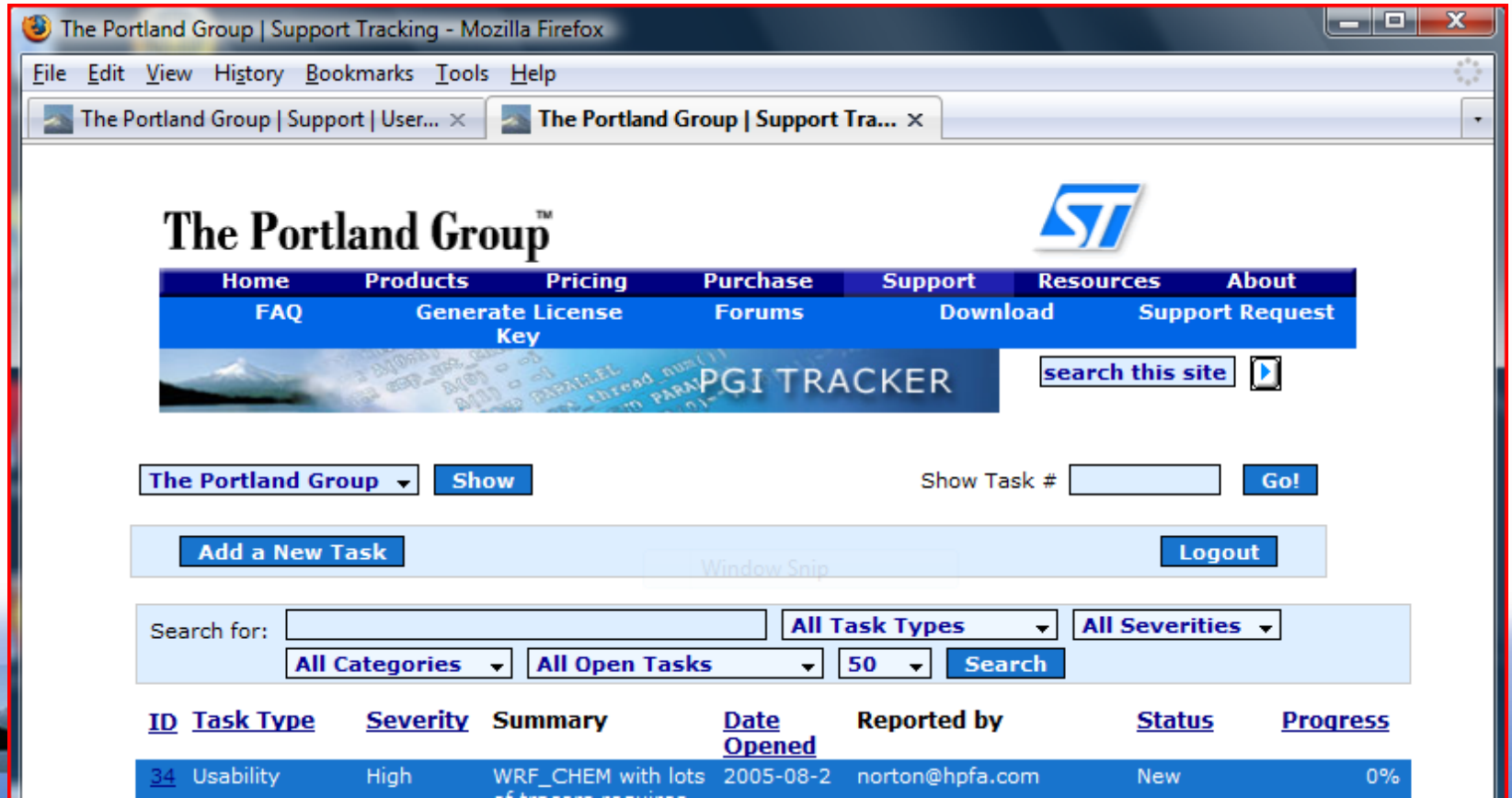
The Quick Start Seminar is an on-site ½ day introduction to PGI compilers and tools intended to cover best practices issues for getting code up and running optimally, and giving correct results, in the shortest amount of time.

PGI offers additional training ranging from “hands-on interaction with code” sessions to in depth training on customer specific performance profiling, to customer specific application optimization, including assembly language seminars.

Customized training can be incorporated into the Premier Services program as desired by the customer.

The PGI Tracker

The Tracker is a web based support interface which is used to capture dialogue between the customer and engineering and serve as a portal for uploading and download code.



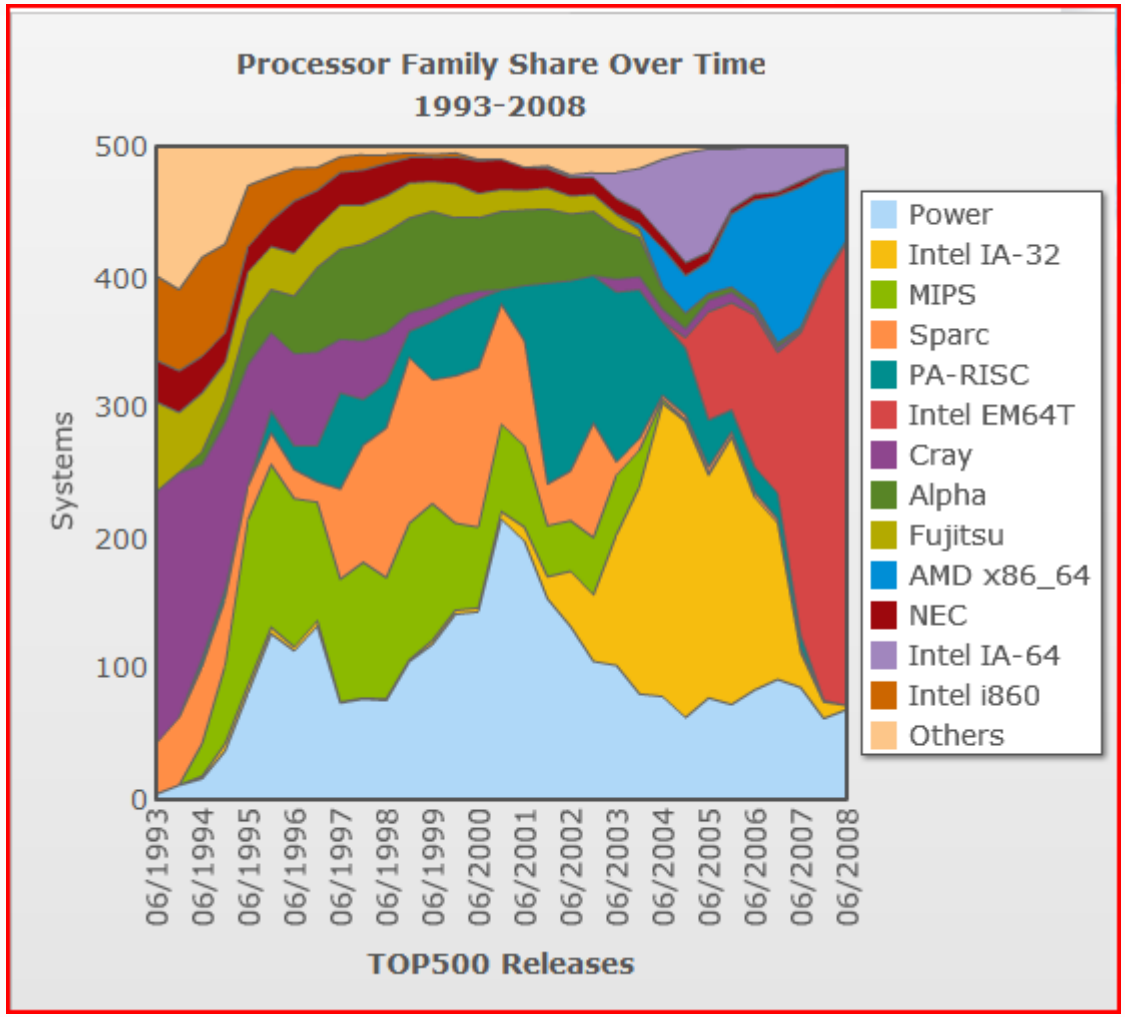
PGI Premier Support Motivation

Customers – especially those with very large systems and specialty applications – have motivation to understand in detail the performance of their codes and have a willingness to include compiler expertise directly on their code development teams.

Code team members who specialize in the science of the application often do not have expertise in how a compiler views their application.

By adding a PGI compiler engineer to the application development team, the team gets access to in depth compiler knowledge, knowledge about how the compiler views code (or should view code) and therefore a team member who can help guide the code development process to optimize application performance while also working on the compiler so that is better understands the code.

TOP 500 CPU Architecture Trends



Data From
top500.org

The Portland Group

Common Performance Challenges

Vectorization on both Intel and AMD processors
- Conflicts with C++ and F90 “ease of use” programming techniques

Multi-core issues
- Memory bandwidth
- MPI, OpenMP, and auto parallelization

IPA – Interproceedural Analysis and Inlining
- IPA and inline enabled libraries

Important PGI Compiler Options

-fast	Includes "**-fast –Mvect=sse -Mcache_align –Mnoframe -Mlre**"
-Mipa=fast	Enable inter-procedural analysis (IPA) and optimization
-Mipa=fast,inline	Enable IPA-based optimization *and* function inlining
-Mpfi … -Mpfo	Enable profile- and data-feedback based optimizations
-Minline	Inline functions and subroutines
-Mconcur	Try to auto-parallelize loops for SMP/Dual-core systems
-mp[=align]	Process OpenMP/SGI directives and pragmas
-mcmodel=medium	Enable data > 2GB on AMD64/EM64T running 64-bit Linux
-Minfo	Compile-time optimization/parallelization messages
-Mneginfo	Compile-time messages indicating what prevented an optimization
-help	Compiler options and usage

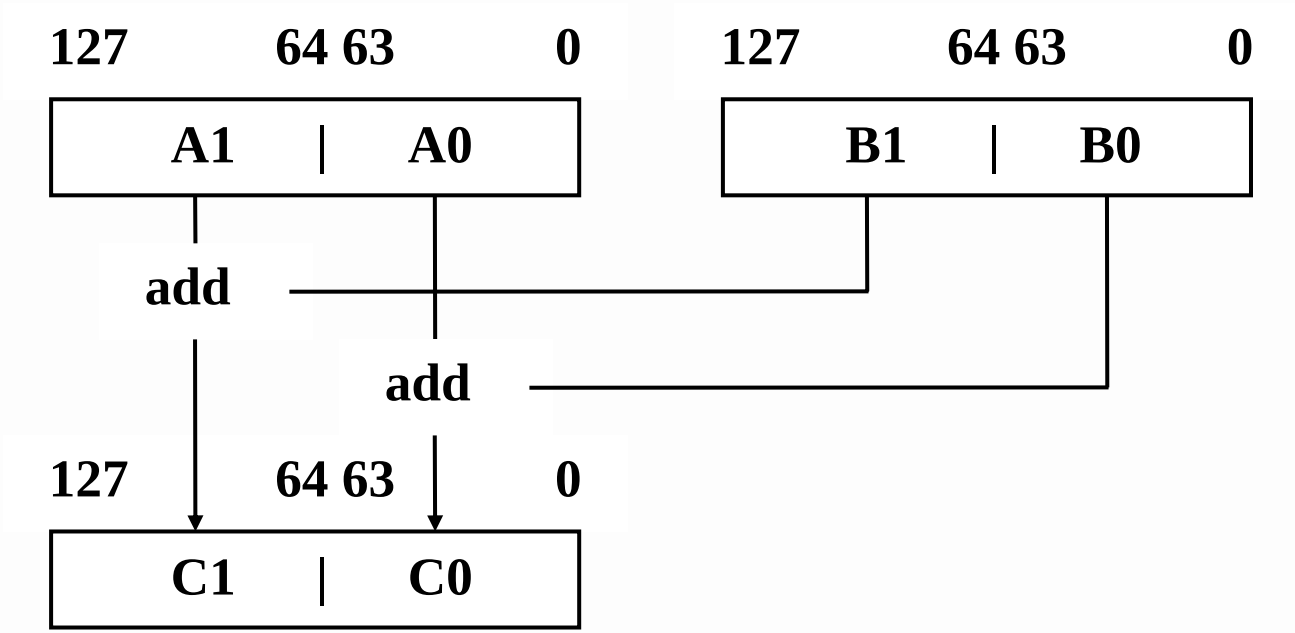
–fast –Mipa=fast usually best for "compile-and-go"

What is Vectorization on x64 CPUs?

- **By a Programmer**: writing or modifying algorithms and loops to enable or maximize generation of x64 packed Streaming SIMD Extensions (SSE) instructions by a vectorizing compiler

- **By a Compiler**: identifying and transforming loops to use packed SSE arithmetic instructions which operate on more than one data element per instruction

Double-precision Packed SSE Operations on x64 CPUs



The Portland Group

Double-precision Packed SSE *Implementations* on x64 CPUs

1st Gen

```
Cycle i:        [A1|A0]   [B1|B0]
                     \         /
                      A0+B0
                      .....
                      .....
Cycle i + 1:    [A1|A0]   [B1|B0]
                     \         /
                      A1+B1
                      A0+B0
                      .....
Cycle i + p - 1:  .....
                  A1+B1
                  A0+B0
                     \
                  [ .. |C0]
Cycle I + p:      .....
                  .....
                  A1+B1
                     /
                  [C1|C0]
```

2nd Gen

```
Cycle i:        [A1|A0]   [B1|B0]
                     \         /
                  [A1+B1|A0+B0]
                  .....
                  .....
Cycle i + 1: .....
                  [A1+B1|A0+B0]
                  .....

Cycle i + p: .....
                  .....
                  [A1+B1|A0+B0]
                     \         /
                      [C1|C0]
```

The Portland Group

Double-precision Packed SSE *Implementations* on x64 CPUs

3rd Gen

Cycle i: **[A3 | A2 | A1 | A0] [B3 | B2 | B1 | B0]**

| | |

[A3 + B3 | A2 + B2 | A1 + B1 | A0 + B0]

.....

.....

Cycle i + 1: .....

[A3+B3 | A2+B2 | A1+B1 | A0+B0]

.....

Cycle i + p: .....

.....

[A3+B3 | A2+B2 | A1+B1 | A0+B0]

****

[C3 | C2 | C1 | C0]

Vectorizable Loop in SPECFP2K FACEREC Data is REAL*4

```
350 !
351 !   Initialize vertex, similarity and coordinate arrays
352 !
353     Do Index = 1, NodeCount
354        IX = MOD (Index - 1, NodesX) + 1
355        IY = ((Index - 1) / NodesX) + 1
356        CoordX (IX, IY) = Position (1) + (IX - 1) * StepX
357        CoordY (IX, IY) = Position (2) + (IY - 1) * StepY
358        JetSim (Index)  = SUM (Graph (:, :, Index) * &
359     &                    GaborTrafo (:, :, CoordX(IX,IY), CoordY(IX,IY)))
360        VertexX (Index) = MOD (Params%Graph%RandomIndex (Index) - 1, NodesX) + 1
361        VertexY (Index) = ((Params%Graph%RandomIndex (Index) - 1) / NodesX) + 1
362     End Do
```

Inner loop at line 358 is vectorizable, can used packed SSE instructions

Use –Minfo to see Which Loops Vectorize

```
% pgf95 -fastsse -Mipa=fast -Minfo -S graphRoutines.f90
…
localmove:
   334, Loop unrolled 1 times (completely unrolled)
   343, Loop unrolled 2 times (completely unrolled)
   358, Generating vector sse code for inner loop
   364, Generating vector sse code for inner loop
        Generating vector sse code for inner loop
   392, Generating vector sse code for inner loop
   423, Generating vector sse code for inner loop
%
```

Scalar SSE:

```
.LB6_668:
# lineno: 358
        movss   -12(%rax),%xmm2
        movss   -4(%rax),%xmm3
        subl    $1,%edx
        mulss   -12(%rcx),%xmm2
        addss   %xmm0,%xmm2
        mulss   -4(%rcx),%xmm3
        movss   -8(%rax),%xmm0
        mulss   -8(%rcx),%xmm0
        addss   %xmm0,%xmm2
        movss   (%rax),%xmm0
        addq    $16,%rax
        addss   %xmm3,%xmm2
        mulss   (%rcx),%xmm0
        addq    $16,%rcx
        testl   %edx,%edx
        addss   %xmm0,%xmm2
        movaps  %xmm2,%xmm0
        jg      .LB6_625
```

Vector SSE:

```
.LB6_1245:
# lineno: 358
        movlps  (%rdx,%rcx),%xmm2
        subl    $8,%eax
        movlps  16(%rcx,%rdx),%xmm3
        prefetcht0  64(%rcx,%rsi)
        prefetcht0  64(%rcx,%rdx)
        movhps  8(%rcx,%rdx),%xmm2
        mulps   (%rsi,%rcx),%xmm2
        movhps  24(%rcx,%rdx),%xmm3
        addps   %xmm2,%xmm0
        mulps   16(%rcx,%rsi),%xmm3
        addq    $32,%rcx
        testl   %eax,%eax
        addps   %xmm3,%xmm0
        jg      .LB6_1245:
```

Facerec Scalar: 104.2 sec
Facerec Vector: 84.3 sec

Vectorizable C Code Fragment?

```
217   void func4(float *u1, float *u2, float *u3, …
      …
221   for (i = -NE+1, p1 = u2-ny, p2 = n2+ny; i < nx+NE-1; i++)
222         u3[i] += clz * (p1[i] + p2[i]);
223   for (i = -NI+1, i < nx+NE-1; i++) {
224         float vdt = v[i] * dt;
225         u3[i] = 2.*u2[i]-u1[i]+vdt*vdt*u3[i];
226   }
```

```
% pgcc –fastsse –Minfo –Mneginfo functions.c
func4:
    221, Loop unrolled 4 times
    221, Loop not vectorized due to data dependency
    223, Loop not vectorized due to data dependency
```

Pointer Arguments Inhibit Vectorization

```
217   void func4(float *u1, float *u2, float *u3, …
      …
221   for (i = -NE+1, p1 = u2-ny, p2 = n2+ny; i < nx+NE-1; i++)
222       u3[i] += clz * (p1[i] + p2[i]);
223   for (i = -NI+1, i < nx+NE-1; i++) {
224       float vdt = v[i] * dt;
225       u3[i] = 2.*u2[i]-u1[i]+vdt*vdt*u3[i];
226   }
```

```
% pgcc –fastsse –Msafeptr –Minfo functions.c
func4:
    221, Generated vector SSE code for inner loop
         Generated 3 prefetch instructions for this loop
    223, Unrolled inner loop 4 times
```

C Constant Inhibits Vectorization

```
217   void func4(float *u1, float *u2, float *u3, …
      …
221   for (i = -NE+1, p1 = u2-ny, p2 = n2+ny; i < nx+NE-1; i++)
222         u3[i] += clz * (p1[i] + p2[i]);
223   for (i = -NI+1, i < nx+NE-1; i++) {
224         float vdt = v[i] * dt;
225         u3[i] = 2.*u2[i]-u1[i]+vdt*vdt*u3[i];
226   }
```

```
% pgcc –fastsse –Msafeptr –Mfcon –Minfo functions.c
func4:
    221, Generated vector SSE code for inner loop
         Generated 3 prefetch instructions for this loop
    223, Generated vector SSE code for inner loop
         Generated 4 prefetch instructions for this loop
```

-Msafeptr Option and Pragma (sledgehammer vs. scalpel)

–M[no]safeptr[=all | arg | auto | dummy | local | static | global]

all	All pointers are safe
arg	Argument pointers are safe
local	local pointers are safe
static	static local pointers are safe
global	global pointers are safe

#pragma [*scope*] [no]safeptr={arg | local | global | static | all},…

Where *scope* is *global*, *routine* or *loop*

Common Barriers to SSE Vectorization

- **Potential Dependencies & C Pointers** – Give compiler more info with –Msafeptr, pragmas, or **restrict** type qualifer
- **Function Calls** – Try inlining with –Minline or –Mipa=inline
- **Type conversions –** manually convert constants or use flags
- **Large Number of Statements –** Try –Mvect=nosizelimit
- **Too few iterations** – Usually better to unroll the loop
- **Real dependencies** – Must restructure loop, if possible

Barriers to Efficient Execution of Vector SSE Loops

- Not enough work – vectors are too short
- Vectors not aligned to a cache line boundary
- Non unity strides
- Code bloat if altcode is generated

Alegra Loop - Multiple Inhibitors to Vectorization

```
for (i=ell.is; i<ell.ie; ++i, edata+=...) {
   . . .
   double ar = 2.0*tvol/sqrt(colon);
   . . .
   {
   double sound_speed = sqrt(gxgm1*edata[ENER]);
   double local_ts = ar/sound_speed;
   if (local_ts/min_ts < 0.9999) min_ts=local_ts;
   edata[AVmPR] = - gamma_minus_one*edata[DENS]
                  * edata[ENER];
   if (tr_deformation_rate < 0.0) {
      edata[AVmPR] += edata[DENS] * ar
       * tr_deformation_rate
       * (linear * sound_speed – quadratic * ar
          * tr_deformation_rate);
   }
   }
}
```

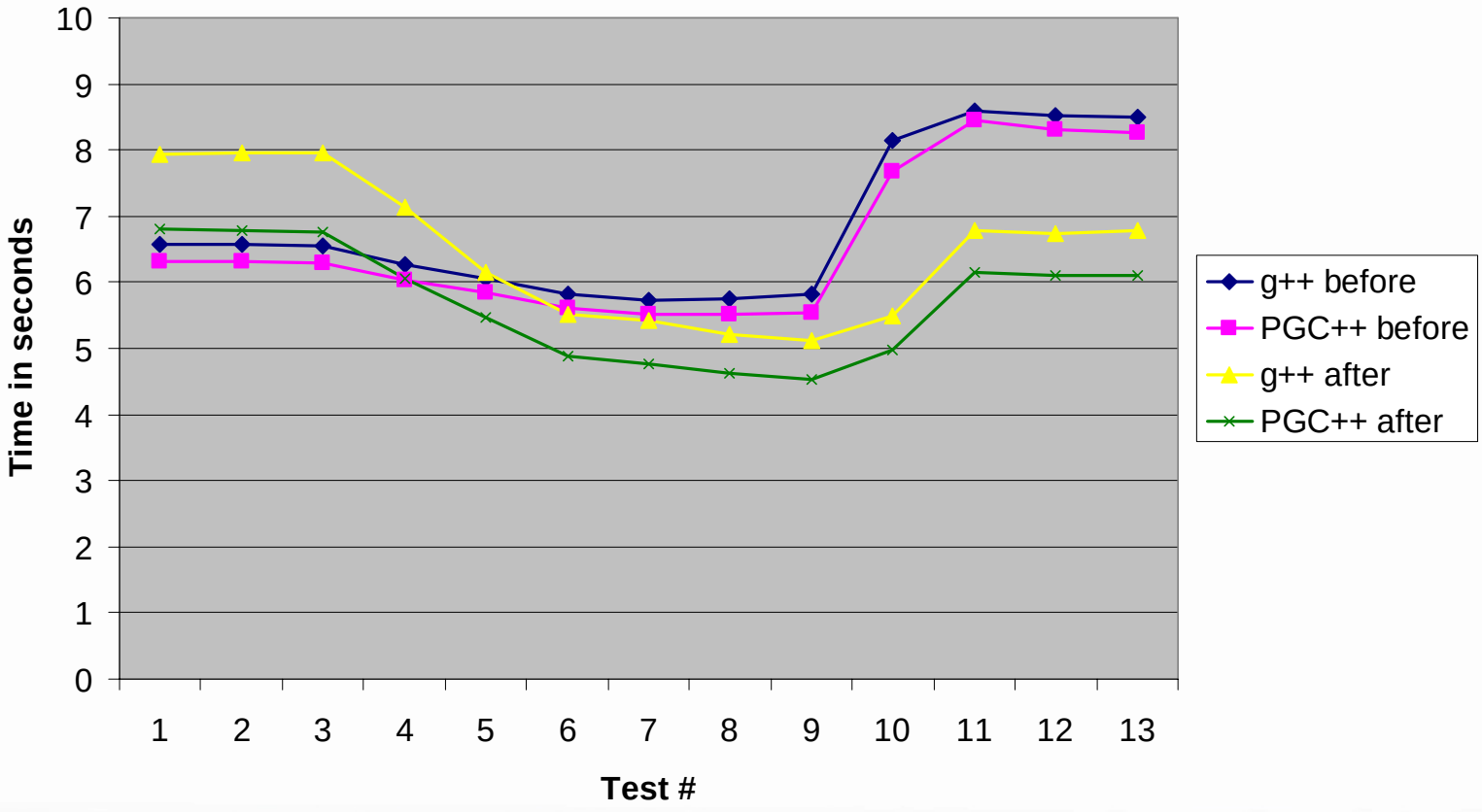
Alegra – Restructuring C++ for Vectorization

- Converted array of structs to vectors of acceleration, velocity, force, etc
- Enabled loop-carried redundancy elimination (LRE) on stencil operations
- Added `__restrict` qualifiers to pointer declarations where safe to do so
- Re-wrote a few key loops to enable vectorization

Alegra - Loop Re-written for Vectorization

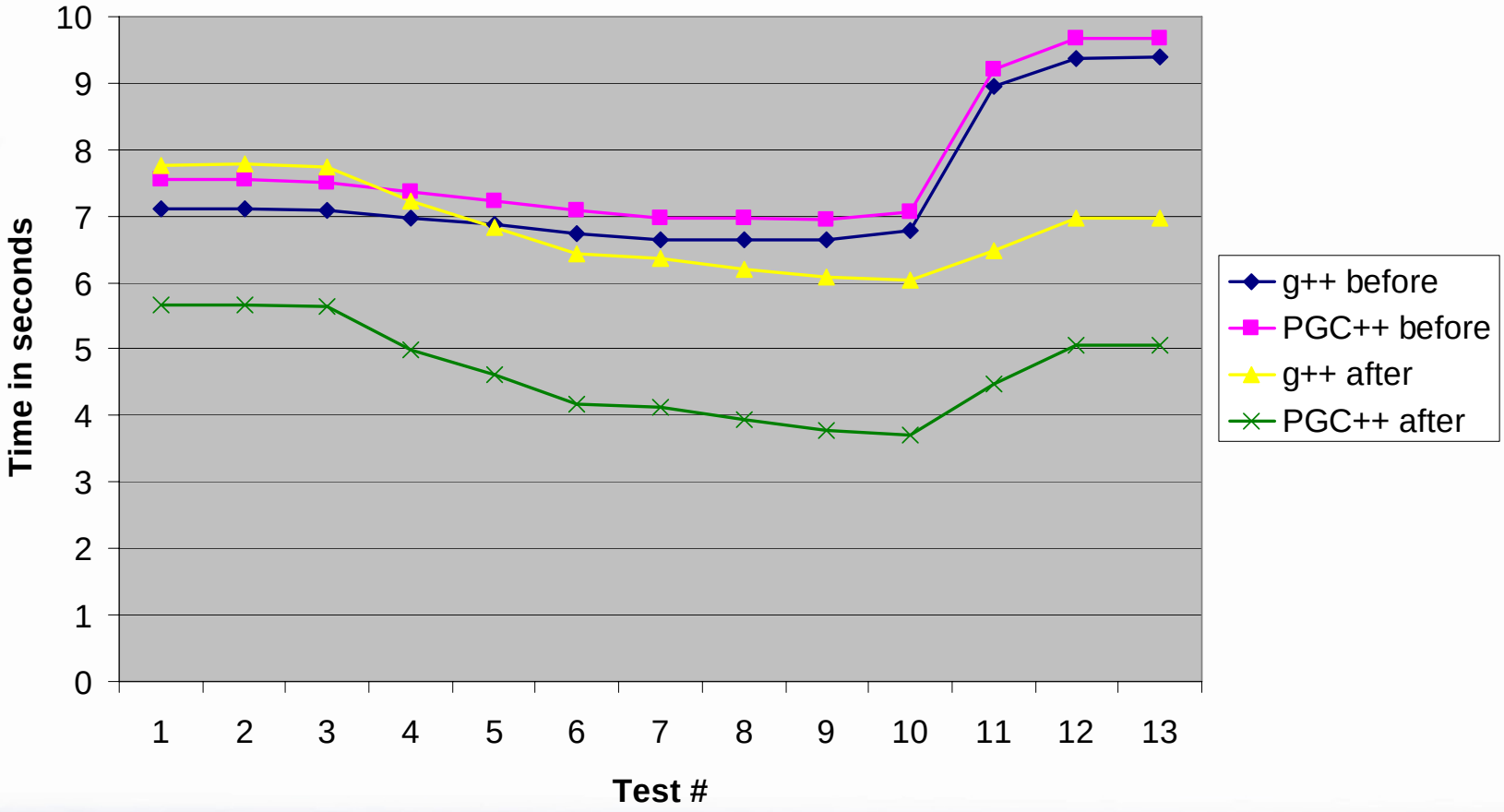
```
. . .
for (i=ell.is; i<ell.ie; ++i) {
   double ar = 2.0*PedataV[i]/sqrt(PedataA[i]);
   double sound_speed = sqrt(gxgm1*PedataE[i]);
   if (ar/sound_speed < local_ts)
      local_ts = ar/sound_speed;
   PedataA[i] = - gamma_minus_one
                * PedataD[i] * PedataE[i];
   double tdrate = (tr_deformation_rate < 0.0) ?
      tr_deformation_rate : 0.0;
   tdrate *= ar;
   PedataA[i] += PedataD[i] * tdrate
                      * (linear * sound_speed
                           - quadratic * tdrate);
}
if (local_ts/min_ts < 0.9999) min_ts=local_ts;
```

1st Generation x64 - AMD Opteron Alegra Kernel Performance



The Portland Group

2nd Generation x64 - Intel Core 2 Alegra Kernel Performance



The Portland Group

Alegra – C++ Challenges

```
for (m = 0; m < mat_max; ++m) {
   ...
   for (b = 0; b < mesh->Num_Element_Blocks(); b++) {
      Element *el;
      LOCAL_ELEMENT_LOOP(el,b) {
         unsigned int element biped = __e_i1__.BitPad();
         if ( element.bitpad & CHANGED) {
--->        Real  volume_old        = el->Volume_Fraction(m);
--->        Real* scratch           = el->Scalar_Array(REMAP_SCRATCH);
--->        int   state             = 0;
--->        int   scratch_pos       = 1;
--->        Material_Data* pmat_data = el->Material_Data_Ptr(m);
            ...
         }   // end if Affected_Element()  i.e. element is remapped
      }    // end     LOCAL_ELEMENT_LOOP
      GHOST_ELEMENT_LOOP(el, b) {
         el->Scalar_Array(REMAP_SCRATCH)[0] = 0.0;
      }
   }   // end for (mesh->Num_Element_Blocks())
}
```

Alegra – C++ Challenges (con't)

For this dataset, the value of mat_max is 21, and the number of element blocks(mesh->Num_Element_Blocks()) is 1. The LOCAL_ELEMENT_LOOP is excuted 160000 times.

Using the debugger, the three lines of code:

1) **Real volume_old = el->Volume_Fraction(m);**

The assembly instructions generated for this line of code dereferenced memory 4 times as follows:

```
movq    160(%rcx), %rdx      <--- Address of el.material_data
movq    (%rdx,%rax,8), %rsi  <--- Address of el.material_data[m].material
movq    40(%rsi), %rax       <--- Address of el.material_data[m].material.data[m']
movl    (%rax), %xmm0        <--- Value of volume_old
```

Alegra – C++ Challenges (con't)

2) **Real* scratch = el->Scalar_Array(REMAP_SCRATCH);**

The assembly instructions generated for this line of code dereferenced memory 2 times as follows:

```
movq    8(%rdx,%rcx), %rsi   <--- Address of el.data
leaq    (%rsi,%rax,8), %rdi  <--- Address of el.data+m
```

3) **Material_Data* pmat_data = el->Material_Data_Ptr(m);**

The assembly instructions generated for this line of code dereferenced memory 2 times as follows:

```
movq    160(%rcx), %r9       <--- Address of el.material_data
movq    (%r9,%r8,8), %rax    <--- Address of el.material_data[m]
```

- **Vectorization** – packed SSE instructions maximize performance
- **Interprocedural Analysis (IPA)** – use it! motivating example
- **Function Inlining –** especially important for C and C++
- **Parallelization** – for multi-core processors
- **Miscellaneous Optimizations** – hit or miss, but worth a try

What can Interprocedural Analysis and Optimization with –Mipa do for You?

- Interprocedural constant propagation
- Pointer disambiguation
- Alignment detection, Alignment propagation
- Global variable mod/ref detection
- F90 shape propagation
- Function inlining
- IPA optimization of libraries, including inlining

Effect of IPA on the WUPWISE Benchmark

PGF95 Compiler Options	Execution Time in Seconds
–fast	156.49
–fast –Mipa=fast	121.65
–fast –Mipa=fast,inline	91.72

- –Mipa=fast => constant propagation => compiler sees complex matrices are all 4x3 => completely unrolls loops
- –Mipa=fast,inline => small matrix multiplies are all inlined

Using Interprocedural Analysis

- Must be used at both compile time and link time
- Non-disruptive to development process – edit/build/run
- Speed-ups of 5% - 10% are common
- –Mipa=safe:<*name*> - safe to optimize functions which call or are called from unknown function/library *name*
- –Mipa=libopt – perform IPA optimizations on libraries
- –Mipa=libinline – perform IPA inlining from libraries

- **Vectorization** – packed SSE instructions maximize performance
- **Interprocedural Analysis (IPA)** – use it! motivating examples
- **Function Inlining –** especially important for C and C++
- **SMP Parallelization** – for multi-core processors
- **Miscellaneous Optimizations** – hit or miss, but worth a try

Explicit Function Inlining

–Minline[=[lib:]<inlib> | [name:]<func> | except:<func> | size:<n> | levels:<n>]

[lib:]<inlib>	Inline extracted functions from *inlib*
[name:]<func>	Inline function func
except:<func>	Do not inline function func
size:<n>	Inline only functions smaller than n statements (approximate)
levels:<n>	Inline n levels of functions

For C++ Codes, PGI Recommends IPA-based inlining or –Minline=levels:10!

Other C++ recommendations

- **Encapsulation, Data Hiding - ** small functions, inline!
- **Exception Handling –** use –no_exceptions until 7.0
- **Overloaded operators, overloaded functions -** okay
- **Pointer Chasing -** -Msafeptr, restrict qualifer, 32 bits?
- **Templates, Generic Programming –** now okay
- **Inheritance, polymorphism, virtual functions –** runtime lookup or check, no inlining, potential performance penalties

- **Vectorization** – packed SSE instructions maximize performance
- **Interprocedural Analysis (IPA)** – use it! motivating examples
- **Function Inlining –** especially important for C and C++
- **SMP Parallelization** – for SMP and multi-core processors
- **Miscellaneous Optimizations** – hit or miss, but worth a try

SMP Parallelization

- **–Mconcur for auto-parallelization on multi-core**
  - Compiler strives for parallel outer loops, vector SSE inner loops
  - –Mconcur=innermost forces a vector/parallel innermost loop
  - –Mconcur=cncall enables parallelization of loops with calls
- **–mp to enable OpenMP parallel programming model**
  - OpenMP programs compiled w/out –mp “just work”
  - Starting in 7.0, two options for idle policy
- **–Mconcur and –mp can be used together!**

MGRID Benchmark Main Loop

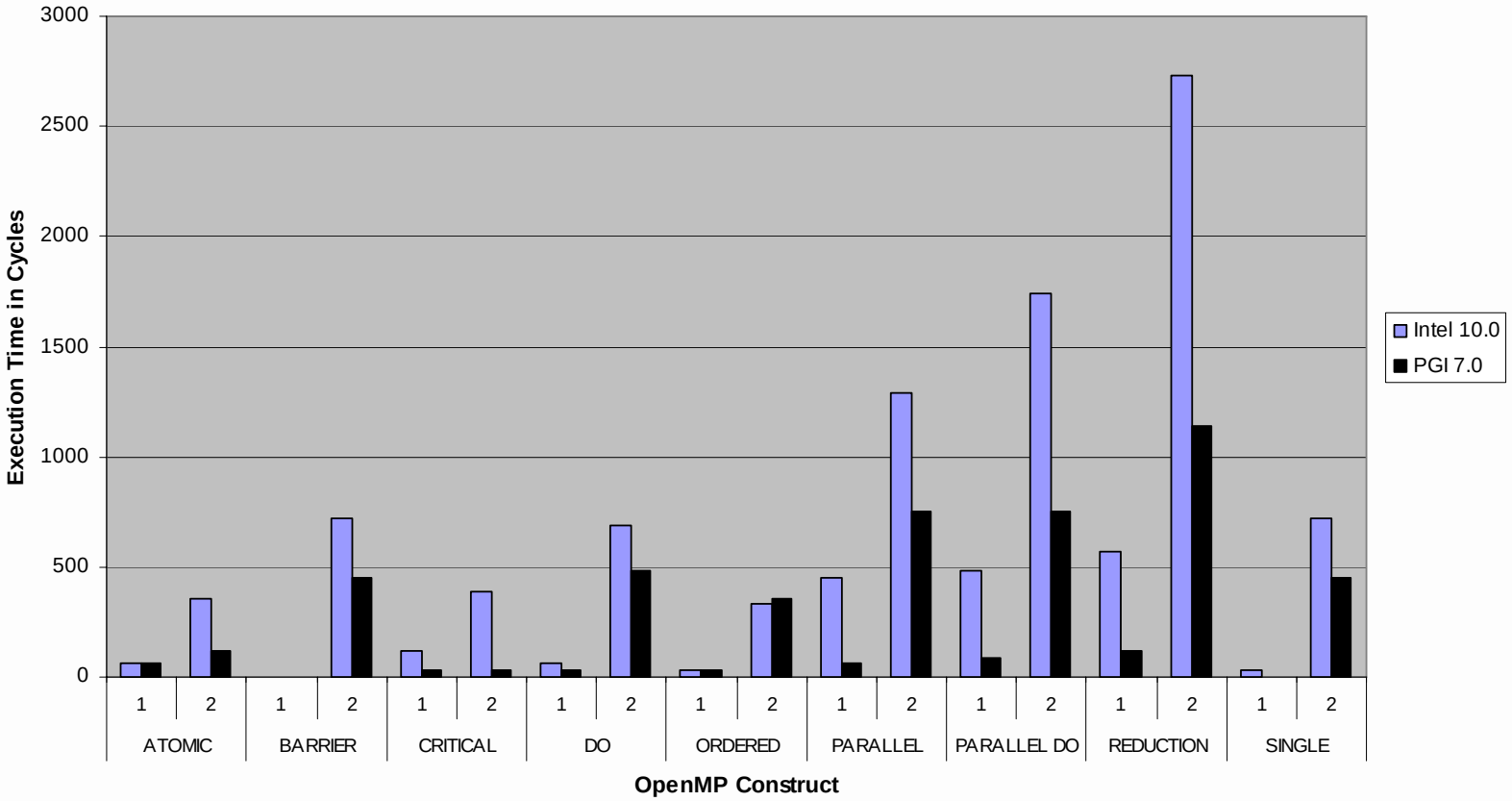
```
      DO 10 I3=2, N-1
      DO 10 I2=2,N-1
      DO 10 I1=2,N-1
10          R(I1,I2,I3) = V(I1,I2,I3)
     &                  -A(0)*(U(I1,I2,I3))
     &                  -A(1)*(U(I1-1,I2,I3)+U(I1+1,I2,I3)
     &                        +U(I1,I2-1,I3)+U(I1,I2+1,I3)
     &                        +U(I1,I2,I3-1)+U(I1,I2,I3+1))
     &                  -A(2)*(U(I1-1,I2-1,I3)+U(I1+1,I2-1,I3)
     &                        +U(I1-1,I2+1,I3)+U(I1+1,I2+1,I3)
     &                        +U(I1,I2-1,I3-1)+U(I1,I2+1,I3-1)
     &                        +U(I1,I2-1,I3+1)+U(I1,I2+1,I3+1)
     &                        +U(I1-1,I2,I3-1)+U(I1-1,I2,I3+1)
     &                        +U(I1+1,I2,I3-1)+U(I1+1,I2,I3+1) )
     &                  -A(3)*(U(I1-1,I2-1,I3-1)+U(I1+1,I2-1,I3-1)
     &                        +U(I1-1,I2+1,I3-1)+U(I1+1,I2+1,I3-1)
     &                        +U(I1-1,I2-1,I3+1)+U(I1+1,I2-1,I3+1)
     &                        +U(I1-1,I2+1,I3+1)+U(I1+1,I2+1,I3+1))
```

Auto-parallel MGRID Overall Speed-up is 40% on Dual-core AMD Opteron

```
% pgf95 –fast –Mipa=fast,inline –Minfo –Mconcur mgrid.f
resid:
. . .
    189, Parallel code for non-innermost loop activated
             if loop count >= 33; block distribution
    291, 4 loop-carried redundant expressions removed
             with 12 operations and 16 arrays
         Generated vector SSE code for inner loop
         Generated 8 prefetch instructions for this loop
         Generated vector SSE code for inner loop
         Generated 8 prefetch instructions for this loop
```

Low-overhead Multi-core Parallelization

Quad-core 2.66Ghz Intel Clovertown, SuSE 10.1
PGI 7.0 and Intel 10.0 Options: -fast +OpenMP



*As measured by the EPCC 2.0 OpenMP Microbenchmarks

The Portland Group

- **Vectorization** – packed SSE instructions maximize performance
- **Interprocedural Analysis (IPA)** – use it! motivating examples
- **Function Inlining –** especially important for C and C++
- **SMP Parallelization** – for multi-core processors
- **Miscellaneous Optimizations** – hit or miss, but worth a try

Miscellaneous Optimizations (1)

- –Mfprelaxed – single-precision sqrt, rsqrt, div performed using reduced-precision reciprocal approximation
- –lacml and –lacml_mp – link in the AMD Core Math Library
- –Mprefetch=d:<p>,n:<q> – control prefetching distance, max number of prefetch instructions per loop
- –tp k8-32 – can result in big performance win on some C/C++ codes that don't require > 2GB addressing; pointer and long data become 32-bits

Miscellaneous Optimizations (2)

- –O3 – more aggressive hoisting and scalar replacement; not part of –fastsse, always time your code to make sure it's faster
- For C++ codes: ––no_exceptions –zc_eh
- –M[no]movnt – disable / force non-temporal moves
- –V[version] to switch between PGI releases at file level
- –Mvect=noaltcode – disable multiple versions of loops

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

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

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

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

$$offsets = [0, 5, 10, 13, 14]$$

```
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

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

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

```
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), in MFlops **per core**

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

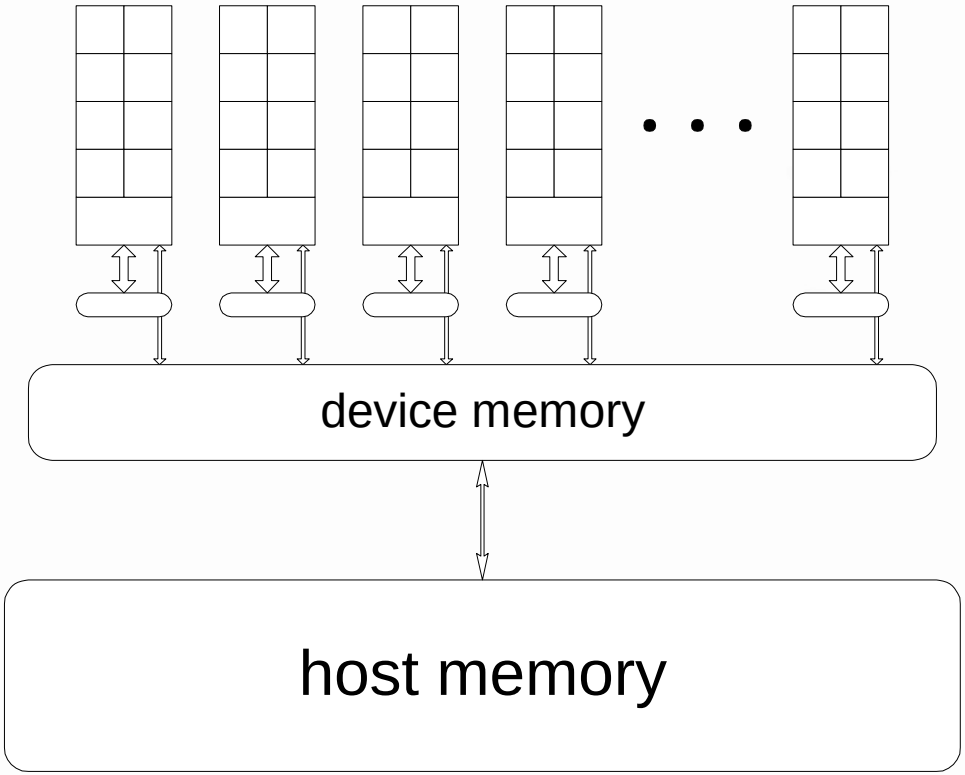
4 cores: Problem Size	24**3	48**3	72**3	100**3
g++ original code, CRS	314	275	270	271
g++ original code, SDS	442	125	125	128
pgCC original code, CRS	291	271	263	264
pgCC original code, SDS	438	125	126	129
pgCC SDS, + restrict	177	149	135	133
g++, SDS, restrict + Strip-mine	469	353	349	349
pgCC SDS, restrict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

PGI Unified Binary™ Technology

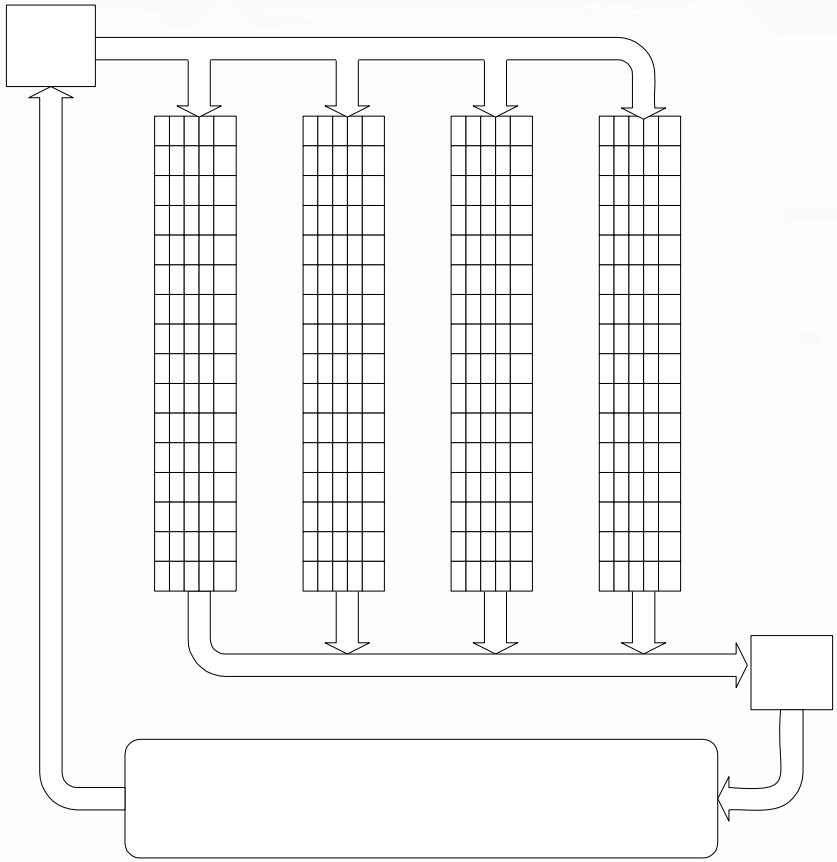
- A single x64 binary with optimized code sequences for both AMD64 and Intel 64 (Intel Core 2)
  - Soon to also include support for different GPGPUs
- Protects customers from the churn of AMD and Intel CPU technology – exploit their innovations without losing binary compatibility
- Reduces SW development, tuning, manufacturing, and maintenance costs

Extending Host-side x64 Compilers to Enable Incremental use of GPGPUs

NVIDIA Architecture

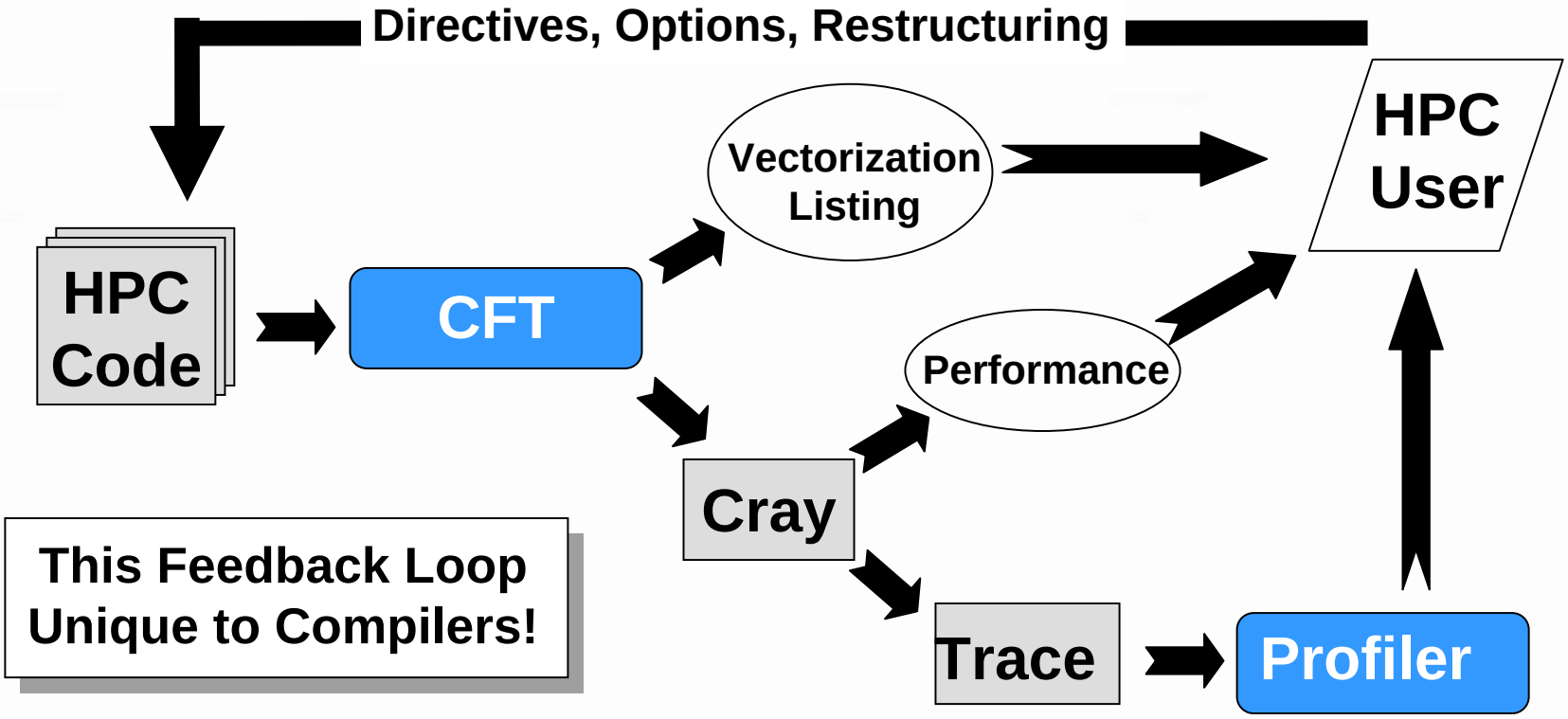


AMD/ATI Architecture

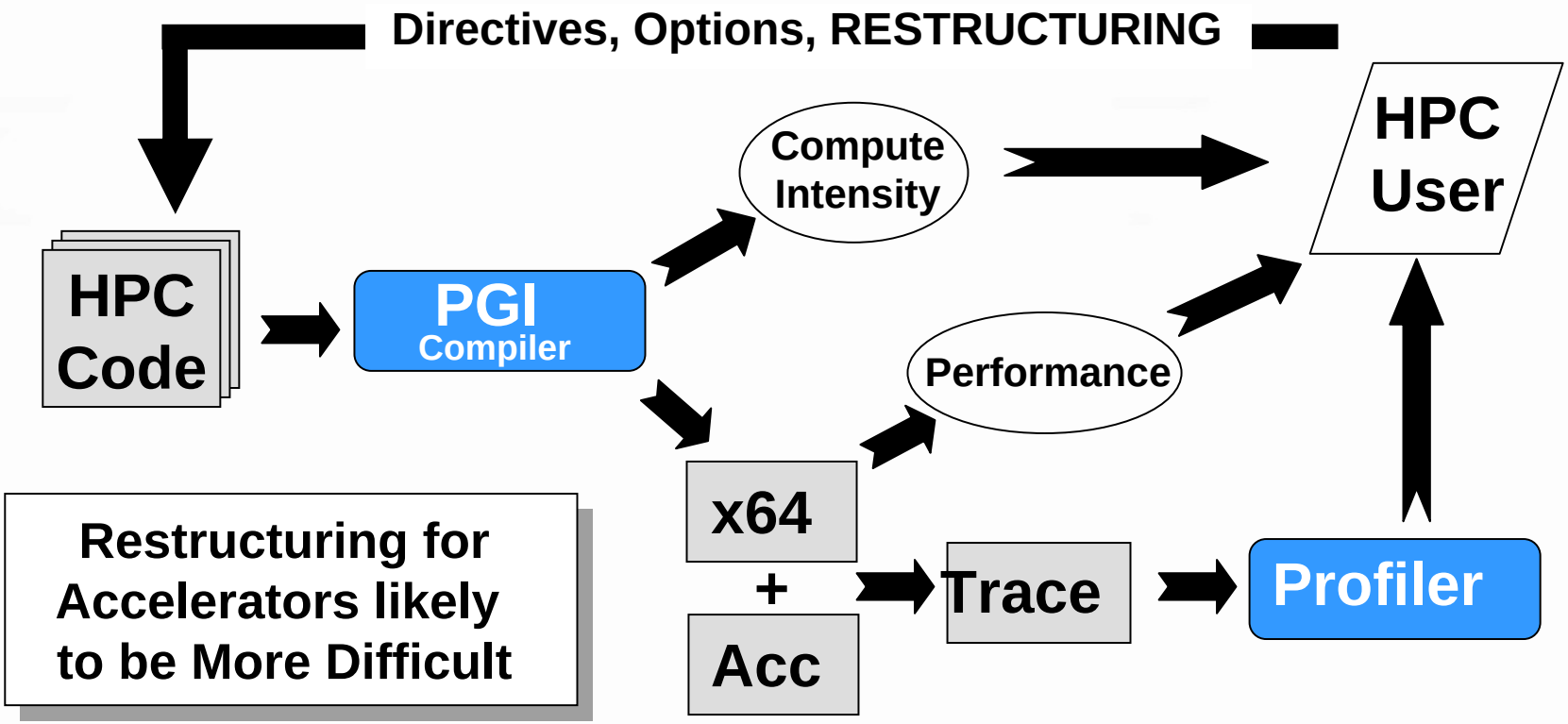


How did we make Vectors Work?

Compiler-to-Programmer Feedback – a classic "Virtuous Cycle"



Compiler-to-Programmer Feedback can do Same for Accelerators



GPGPU-enabled PGI Compilers

```
      SUBROUTINE SAXPY (A,X,Y,N)
      INTEGER N
      REAL A,X(N),Y(N)
!$ACC REGION
      DO I = 1, N
         X(I) = A*X(I) + Y(I)
      ENDDO
!$ACC END REGION
      END
```

compile

Host x64 asm File

```
saxpy_:
        …
        movl    (%rbx), %eax
        movl    %eax, -4(%rbp)
        call    __NVInit
        . . .
        call    __NVRegisterFunction
        …
        call    __NVAlloc
        …
        call    __NVUpload
        …
        call    __NVArgument
        …
        Call    __NVCall
        …
```

+

Auto-generated CUDA

```
typedef struct dim3{ unsigned int x,y,z; }dim3;
typedef struct uint3{ unsigned int x,y,z; }uint3;
extern uint3 const threadIdx, blockIdx;
extern dim3 const blockDim, gridDim;
static __attribute__((__global__)) void
pgicuda(
    __attribute__((__shared__)) int tc,
    __attribute__((__shared__)) int i1,
    __attribute__((__shared__)) int i2,
    __attribute__((__shared__)) int _n,
    __attribute__((__shared__)) float* _c,
    __attribute__((__shared__)) float* _b,
    __attribute__((__shared__)) float* _a )
{  int i;  int p1; int _i;
    i = blockIdx.x * 64 + threadIdx.x;
    if( i < tc ){
        _a[i+i2-1] = ((_c[i+i2-1]+_c[i+i2-1])+_b[i+i2-1]);
        _b[i+i2-1] = _c[i+i2];
        _i = (_i+1);
        p1 = (p1-1);
    } }
```

link

Unified a.out

execute … with no change to existing makefiles, scripts, programming environment, etc

The Portland Group

HPC Processors - Situation Analysis

- **Clock rates trumped architecture for 15 years**
- **Clock rates stall out => Architecture is back!**
- **Relaxed binary compatibility constraints of HPC**
- **Accelerator space wide open for rapid innovation of well-designed, integrated parallel architectures**

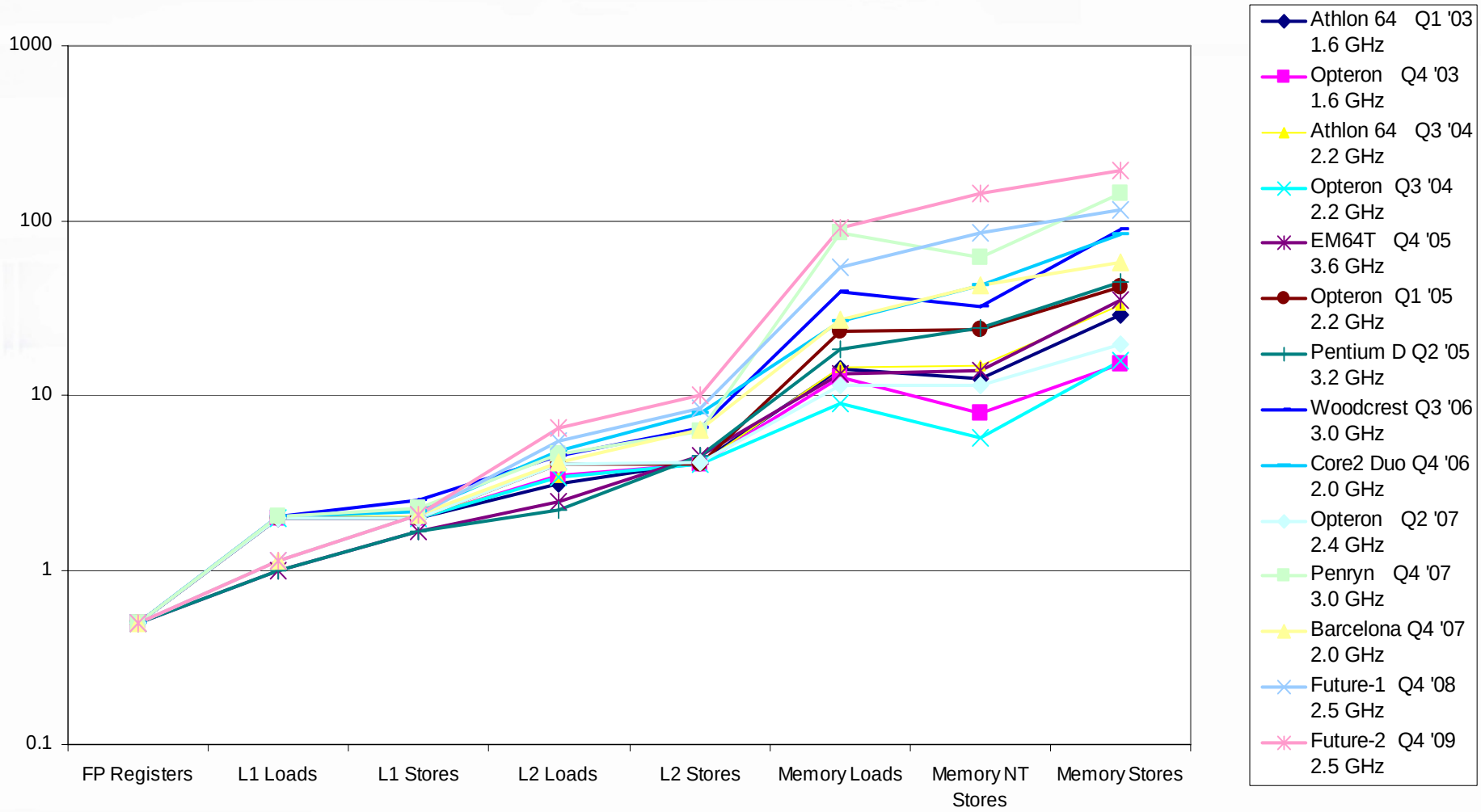
How to make Accelerators accessible to the HPC masses?!?

Accelerator Programming Today - Difficult at Best

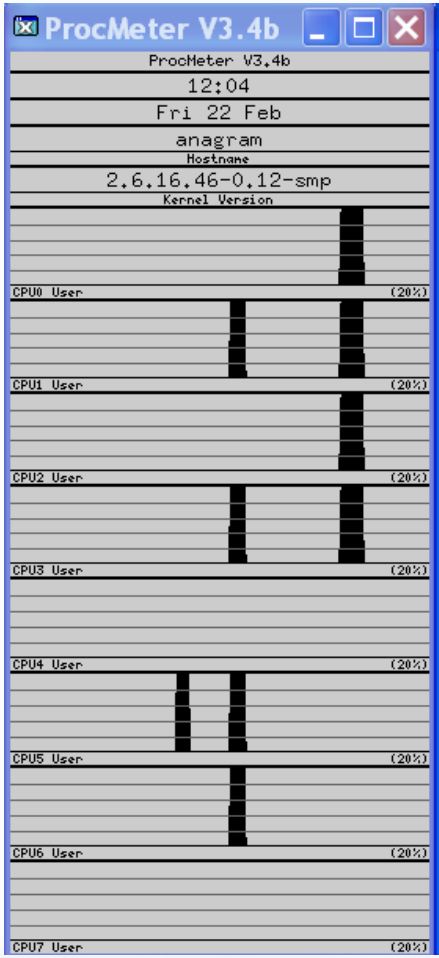
- **Avoid the problem, use pre-packaged Applications**
  - **AMBER (Clearspeed), SPICE (nVidia), NAMD (multiple), MatLab/STAR-P**
- **Use pre-packaged Libraries (BLAS, LAPACK, etc)**
- **C++ Class Libraries, run-time code generation**
- **C-like languages – CUDA, Brook+, Cn, etc**

Why can't we just extend host-side Fortran, C, C++ with directives and pragmas?

Compute Intensity Required to Sustain Floating-Point Speed



Standardizing Quad-Core Performance




```
brentl@suse93:~/simple
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>
```

Compute Intensity & Potential Performance

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

$$GFLOPS = Intensity \times Bandwidth$$

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

Programming Model Considerations - Directives/pragmas and Challenges

- Vector
  - !dir$ ivdep, assert(n>0), shortvector
  - strip mining, vectorizing non-inner loops, conditionals, calls, maximizing stride 1 accesses, data alignment
- OpenMP
  - !$omp parallel / endparallel, #pragma omp parallel, !$omp do, #pragma omp for, private(data), firstprivate(data), nowait, !$omp barrier
  - efficient thread management & synchronization, minimizing overhead, nested parallelism, load balancing, caches
- Accelerator
  - begin / end, define region to move efficiently, data management, local allocation, move to/from device
  - compute intensity / profitability analysis, loop mappings to different levels of device parallelism, efficient use of data bandwidth, minimizing data movement, which ones to target

PGI Accelerator Programming

- Minimal changes to source code
  - No language changes (directives)
  - Minimal library call requirements
- Performance feedback to programmer
- no changes to makefiles, build process, other tools

Accelerator Directives

```
!$acc region
   do i = 1, n
     do j = 1, m
       a(i,j) = 0.0
       do k = 1,p
         a(i,j) = a(i,j) + b(i,k)*c(k,j)
       enddo
     enddo
   enddo
!$acc end region
```

Accelerator Directives

- !$acc region ... !$acc end region
- !$acc private(a)
- !$acc copy(a)
- !$acc map([gang,thread,simd])
- !$acc stripmine(n)

The Weather Research and Forecast Model (WRF) as a Validating Application

- One of the most important app's in one of the most important segments of the HPC market
- Distributed in source form
- Approximately 3K registered users
- About 400K lines of Fortran 90
- Nearly 100% of compute time in 10 – 12 kernels
- Parallelism well-suited to migrate all kernels to GPUs
- WSM5 kernel consumes 25% of compute time …

WSM5 2D kernel

```
!$acc region
      DO j=jts,jte
        pi = 4. * atan(1.)
        do k = kts, kte
          do i = its, ite
            qc(i,k,j) = max(qc(i,k,j),0.0)
            qr(i,k,j) = max(qr(i,k,j),0.0)
            qi(i,k,j) = max(qi(i,k,j),0.0)
            qs(i,k,j) = max(qs(i,k,j),0.0)
          enddo
        enddo
        do k = kts, kte
          do i = its, ite
            cpm(i,k) = cpmcal(q(i,k,j))
            xl(i,k) = xlcal(t(i,k,j))
          enddo
        enddo
        loops = max(int(delt/dtcldcr),1)
        dtcld = delt/loops
        if(delt.le.dtcldcr) dtcld = delt
        do i = its, ite
          mstep(i) = 1
          flgcld(i) = .true.
        enddo
        do k = kts, kte
          do i = its, ite
            tvec0 = 1/den(i,k,j)
            tvec0 = tvec0*den0
            denfac(i,k) = sqrt(tvec0)
          enddo
        enddo
```

```
hsub = xls
hvap = xlv0
cvap = cpv
ttp=t0c+0.01
dldt=cvap-cliq
xa=-dldt/rv
xb=xa+hvap/(rv*ttp)
dldti=cvap-cice
xai=-dldti/rv
xbi=xai+hsub/(rv*ttp)
do k = kts, kte
  do i = its, ite
    tr=ttp/t(i,k,j)
    lqs(i,k,1)=psat*exp(log(tr)*(xa))*exp(xb*(1.-tr))
    lqs(i,k,1) = ep2 * lqs(i,k,1) / (p(i,k,j) - lqs(i,k,1))
    lqs(i,k,1) = max(lqs(i,k,1),qmin)
    rh(i,k,1) = max(q(i,k,j) / lqs(i,k,1),qmin)
    if(t(i,k,j).lt.ttp) then
     lqs(i,k,2)=psat*exp(log(tr)*(xai))*exp(xbi*(1.-tr))
    else
      lqs(i,k,2)=psat*exp(log(tr)*(xa))*exp(xb*(1.-tr))
    endif
    lqs(i,k,2) = ep2 * lqs(i,k,2) / (p(i,k,j) - lqs(i,k,2))
    lqs(i,k,2) = max(lqs(i,k,2),qmin)
    rh(i,k,2) = max(q(i,k,j) / lqs(i,k,2),qmin)
  enddo
enddo
do k = kts, kte
  do i = its, ite
    prevp(i,k) = 0.
    psdep(i,k) = 0.
```

The Portland Group

```
      praut(i,k) = 0.
      psaut(i,k) = 0.
      pracw(i,k) = 0.
      psaci(i,k) = 0.
      psacw(i,k) = 0.
      pigen(i,k) = 0.
      pidep(i,k) = 0.
      pcond(i,k) = 0.
      psmlt(i,k) = 0.
      psevp(i,k) = 0.
      falk(i,k,1) = 0.
      falk(i,k,2) = 0.
      fall(i,k,1) = 0.
      fall(i,k,2) = 0.
      fallc(i,k) = 0.
      falkc(i,k) = 0.
      xni(i,k) = 1.e3
    enddo
  enddo
  do k = kts, kte
    do i = its, ite
      supcol = t0c-t(i,k,j)
      n0sfac(i,k) = max(min(exp(alpha*supcol),n0smax/n0s),1.)
      if(qr(i,k,j).le.qcrmin)then
        rslope(i,k,1) = rslopermax
        rslopeb(i,k,1) = rsloperbmax
        rslope2(i,k,1) = rsloper2max
        rslope3(i,k,1) = rsloper3max
      else
        rslope(i,k,1) = 1./lamdar(qr(i,k,j),den(i,k,j))
        rslopeb(i,k,1) = exp(log(rslope(i,k,1))*(bvtr))
        rslope2(i,k,1) = rslope(i,k,1)*rslope(i,k,1)
        rslope3(i,k,1) = rslope2(i,k,1)*rslope(i,k,1)
      endif
      if(qs(i,k,j).le.qcrmin)then
        rslope(i,k,2) = rslopesmax
        rslopeb(i,k,2) = rslopesbmax
        rslope2(i,k,2) = rslopes2max
        rslope3(i,k,2) = rslopes3max
      else
        rslope(i,k,2) = 1./lamdas(qs(i,k,j),den(i,k,j),n0sfac(i,k))
        rslopeb(i,k,2) = exp(log(rslope(i,k,2))*(bvts))
        rslope2(i,k,2) = rslope(i,k,2)*rslope(i,k,2)
        rslope3(i,k,2) = rslope2(i,k,2)*rslope(i,k,2)
      endif
      temp = (den(i,k,j)*max(qi(i,k,j),qmin))
      temp = sqrt(sqrt(temp*temp*temp))
      xni(i,k) = min(max(5.38e7*temp,1.e3),1.e6)
    enddo
  enddo
  mstepmax = 1
  numdt = 1
  do k = kte, kts, -1
    do i = its, ite
      work1(i,k,1) = pvtr*rslopeb(i,k,1)*denfac(i,k)/delz(i,k,j)
      work1(i,k,2) = pvts*rslopeb(i,k,2)*denfac(i,k)/delz(i,k,j)
      numdt(i) = max(int(max(work1(i,k,1),work1(i,k,2))*dtcld+.5),1)
      if(numdt(i).ge.mstep(i)) mstep(i) = numdt(i)
    enddo
  enddo
  do i = its, ite
```

```
if(mstepmax.le.mstep(i)) mstepmax = mstep(i)
  rmstep(i) = 1./mstep(i)
enddo
do n = 1, mstepmax
  k = kte
  do i = its, ite
    if(n.le.mstep(i)) then
      falk(i,k,1) = den(i,k,j)*qr(i,k,j)*work1(i,k,1)*rmstep(i)
      falk(i,k,2) = den(i,k,j)*qs(i,k,j)*work1(i,k,2)*rmstep(i)
      fall(i,k,1) = fall(i,k,1)+falk(i,k,1)
      fall(i,k,2) = fall(i,k,2)+falk(i,k,2)
      dtcldden = dtcld/den(i,k,j)
      qr(i,k,j) = max(qr(i,k,j)-falk(i,k,1)*dtcldden,0.)
      qs(i,k,j) = max(qs(i,k,j)-falk(i,k,2)*dtcldden,0.)
    endif
  enddo
  do k = kte-1, kts, -1
    do i = its, ite
      if(n.le.mstep(i)) then
        falk(i,k,1) = den(i,k,j)*qr(i,k,j)*work1(i,k,1)*rmstep(i)
        falk(i,k,2) = den(i,k,j)*qs(i,k,j)*work1(i,k,2)*rmstep(i)
        fall(i,k,1) = fall(i,k,1)+falk(i,k,1)
        fall(i,k,2) = fall(i,k,2)+falk(i,k,2)
        dtcldden = dtcld/den(i,k,j)
        rdelz = 1./delz(i,k,j)
        qr(i,k,j) = max(qr(i,k,j)-(falk(i,k,1)-falk(i,k+1,1) &
                        *delz(i,k+1,j)*rdelz)*dtcldden,0.)
        qs(i,k,j) = max(qs(i,k,j)-(falk(i,k,2)-falk(i,k+1,2) &
                        *delz(i,k+1,j)*rdelz)*dtcldden,0.)
      endif
    enddo
  enddo
  do k = kte, kts, -1
    do i = its, ite
      if(n.le.mstep(i)) then
        if(t(i,k,j).gt.t0c.and.qs(i,k,j).gt.0.) then
          xlf = xlf0
          work2(i,k)= (exp(log(((1.496e-6*((t(i,k,j))*sqrt(t(i,k,j))) &
              /((t(i,k,j))+120.)/(den(i,k,j)))/(8.794e-5 &
              *exp(log(t(i,k,j))*(1.81))/p(i,k,j)))) &
              *((.3333333)))/sqrt((1.496e-6*((t(i,k,j)) &
              *sqrt(t(i,k,j)))/((t(i,k,j))+120.)/(den(i,k,j)))) &
              *sqrt(sqrt(den0/(den(i,k,j)))))
          coeres = rslope2(i,k,2)*sqrt(rslope(i,k,2)*rslopeb(i,k,2))
          psmlt(i,k) = &
              (1.414e3*(1.496e-6 * ((t(i,k,j))*sqrt(t(i,k,j))) /&
              ((t(i,k,j))+120.)/(den(i,k,j)) )*(den(i,k,j)))&
              /xlf*(t0c-t(i,k,j))*pi/2. &
              *n0sfac(i,k)*(precs1*rslope2(i,k,2)+precs2 &
              *work2(i,k)*coeres)
          psmlt(i,k) = min(max(psmlt(i,k)*dtcld/mstep(i), &
              -qs(i,k,j)/mstep(i)),0.)
          qs(i,k,j) = qs(i,k,j) + psmlt(i,k)
          qr(i,k,j) = qr(i,k,j) - psmlt(i,k)
          t(i,k,j) = t(i,k,j) + xlf/cpm(i,k)*psmlt(i,k)
        endif
      endif
    enddo
  enddo
enddo
mstepmax = 1
mstep = 1
numdt = 1
do k = kte, kts, -1
  do i = its, ite
    if(qi(i,k,j).le.0.) then
      work2c(i,k) = 0.
    else
      xmi = den(i,k,j)*qi(i,k,j)/xni(i,k)
      diameter = max(min(dicon * sqrt(xmi),dimax), 1.e-25)
      work1c(i,k) = 1.49e4*exp(log(diameter)*(1.31))
      work2c(i,k) = work1c(i,k)/delz(i,k,j)
    endif
    numdt(i) = max(int(work2c(i,k)*dtcld+.5),1)
```

```
if(numdt(i).ge.mstep(i)) mstep(i) = numdt(i)
  enddo
enddo
do i = its, ite
  if(mstepmax.le.mstep(i)) mstepmax = mstep(i)
enddo
do n = 1, mstepmax
  k = kte
  do i = its, ite
    if(n.le.mstep(i)) then
      falkc(i,k) = den(i,k,j)*qi(i,k,j)*work2c(i,k)/mstep(i)
      fallc(i,k) = fallc(i,k)+falkc(i,k)
      qi(i,k,j) = max(qi(i,k,j)-falkc(i,k)*dtcld/den(i,k,j),0.)
    endif
  enddo
  do k = kte-1, kts, -1
    do i = its, ite
      if(n.le.mstep(i)) then
        falkc(i,k) = den(i,k,j)*qi(i,k,j)*work2c(i,k)/mstep(i)
        fallc(i,k) = fallc(i,k)+falkc(i,k)
        qi(i,k,j) = max(qi(i,k,j)-(falkc(i,k)-falkc(i,k+1) &
                  *delz(i,k+1,j)/delz(i,k,j))*dtcld/den(i,k,j),0.)
      endif
    enddo
  enddo
enddo
do i = its, ite
  fallsum = fall(i,kts,1)+fall(i,kts,2)+fallc(i,kts)
  fallsum_qsi = fall(i,kts,2)+fallc(i,kts)
  rainncv(i,j) = 0.
  if(fallsum.gt.0.) then
    rainncv(i,j) = fallsum*delz(i,kts,j)/denr*dtcld*1000.
    rain(i,j) = fallsum*delz(i,kts,j)/denr*dtcld*1000. + rain(i,j)
  endif
    snowncv(i,j) = 0.
    if(fallsum_qsi.gt.0.) then
      snowncv(i,j) = fallsum_qsi*delz(i,kts,j)/denr*dtcld*1000.
      snow(i,j) = fallsum_qsi*delz(i,kts,j)/denr*dtcld*1000. + snow(i,j)
    endif
  sr(i,j) = 0.
  if(fallsum.gt.0.)sr(i,j)= &
       fallsum_qsi*delz(i,kts,j)/denr*dtcld*1000./(rainncv(i,j)+1.e-12)
enddo
do k = kts, kte
  do i = its, ite
    supcol = t0c-t(i,k,j)
    xlf = xls-xl(i,k)
    if(supcol.lt.0.) xlf = xlf0
    if(supcol.lt.0.and.qi(i,k,j).gt.0.) then
      qc(i,k,j) = qc(i,k,j) + qi(i,k,j)
      t(i,k,j) = t(i,k,j) - xlf/cpm(i,k)*qi(i,k,j)
      qi(i,k,j) = 0.
    endif
    if(supcol.gt.40..and.qc(i,k,j).gt.0.) then
      qi(i,k,j) = qi(i,k,j) + qc(i,k,j)
      t(i,k,j) = t(i,k,j) + xlf/cpm(i,k)*qc(i,k,j)
      qc(i,k,j) = 0.
    endif
    if(supcol.gt.0..and.qc(i,k,j).gt.0.) then
      pfrzdtc = min(pfrz1*(exp(pfrz2*supcol)-1.) &
          *den(i,k,j)/denr/xncr*qc(i,k,j)*qc(i,k,j)*dtcld,qc(i,k,j))
      qi(i,k,j) = qi(i,k,j) + pfrzdtc
      t(i,k,j) = t(i,k,j) + xlf/cpm(i,k)*pfrzdtc
      qc(i,k,j) = qc(i,k,j)-pfrzdtc
    endif
    if(supcol.gt.0..and.qr(i,k,j).gt.0.) then
      temp = rslope(i,k,1)
      temp = temp*temp*temp*temp*temp*temp*temp
      pfrzdtr = min(20.*(pi*pi)*pfrz1*n0r*denr/den(i,k,j) &
                *(exp(pfrz2*supcol)-1.)*temp*dtcld, &
                qr(i,k,j))
      qs(i,k,j) = qs(i,k,j) + pfrzdtr
      t(i,k,j) = t(i,k,j) + xlf/cpm(i,k)*pfrzdtr
      qr(i,k,j) = qr(i,k,j)-pfrzdtr
```

```
    endif
  enddo
enddo
do k = kts, kte
  do i = its, ite
    if(qr(i,k,j).le.qcrmin)then
      rslope(i,k,1) = rslopermax
      rslopeb(i,k,1) = rsloperbmax
      rslope2(i,k,1) = rsloper2max
      rslope3(i,k,1) = rsloper3max
    else
      rslope(i,k,1) = 1./(sqrt(sqrt(pidn0r/((qr(i,k,j))*(den(i,k,j))))))
      rslopeb(i,k,1) = exp(log(rslope(i,k,1))*(bvtr))
      rslope2(i,k,1) = rslope(i,k,1)*rslope(i,k,1)
      rslope3(i,k,1) = rslope2(i,k,1)*rslope(i,k,1)
    endif
    if(qs(i,k,j).le.qcrmin)then
      rslope(i,k,2) = rslopesmax
      rslopeb(i,k,2) = rslopesbmax
      rslope2(i,k,2) = rslopes2max
      rslope3(i,k,2) = rslopes3max
    else
      rslope(i,k,2) = 1./(sqrt(sqrt(pidn0s*(n0sfac(i,k))/ &
                         ((qs(i,k,j))*(den(i,k,j))))))
      rslopeb(i,k,2) = exp(log(rslope(i,k,2))*(bvts))
      rslope2(i,k,2) = rslope(i,k,2)*rslope(i,k,2)
      rslope3(i,k,2) = rslope2(i,k,2)*rslope(i,k,2)
    endif
  enddo
enddo
do k = kts, kte
  do i = its, ite
    work1(i,k,1) = diffac(xl(i,k),p(i,k,j),t(i,k,j),&
                          den(i,k,j),lqs(i,k,1))
    work1(i,k,2) = diffac(xls,p(i,k,j),t(i,k,j),den(i,k,j),lqs(i,k,2))
    work2(i,k) = venfac(p(i,k,j),t(i,k,j),den(i,k,j))
  ENDDO
ENDDO
do k = kts, kte
  do i = its, ite
    supsat = max(q(i,k,j),qmin)-lqs(i,k,1)
    satdt = supsat/dtcld
    if(qc(i,k,j).gt.qc0) then
      praut(i,k) = qck1*exp(log(qc(i,k,j))*((7./3.)))
      praut(i,k) = min(praut(i,k),qc(i,k,j)/dtcld)
    endif
    if(qr(i,k,j).gt.qcrmin.and.qc(i,k,j).gt.qmin) then
      pracw(i,k) = min(pacrr*rslope3(i,k,1)*rslopeb(i,k,1) &
                       *qc(i,k,j)*denfac(i,k),qc(i,k,j)/dtcld)
    endif
    if(qr(i,k,j).gt.0.) then
      coeres = rslope2(i,k,1)*sqrt(rslope(i,k,1)*rslopeb(i,k,1))
      prevp(i,k) = (rh(i,k,1)-1.)*(precr1*rslope2(i,k,1) &
                       +precr2*work2(i,k)*coeres)/work1(i,k,1)
      if(prevp(i,k).lt.0.) then
        prevp(i,k) = max(prevp(i,k),-qr(i,k,j)/dtcld)
        prevp(i,k) = max(prevp(i,k),satdt/2)
      else
        prevp(i,k) = min(prevp(i,k),satdt/2)
      endif
    endif
  enddo
enddo
rdtcld = 1./dtcld
do k = kts, kte
  do i = its, ite
    supcol = t0c-t(i,k,j)
    supsat = max(q(i,k,j),qmin)-lqs(i,k,2)
    satdt = supsat/dtcld
    ifsat = 0
    temp = (den(i,k,j)*max(qi(i,k,j),qmin))
    temp = sqrt(sqrt(temp*temp*temp))
    xni(i,k) = min(max(5.38e7*temp,1.e3),1.e6)
    eacrs = exp(0.07*(-supcol))
```

```
if(supcol.gt.0) then
  if(qs(i,k,j).gt.qcrmin.and.qi(i,k,j).gt.qmin) then
    xmi = den(i,k,j)*qi(i,k,j)/xni(i,k)
    diameter = min(dicon * sqrt(xmi),dimax)
    vt2i = 1.49e4*diameter**1.31
    vt2s = pvts*rslopeb(i,k,2)*denfac(i,k)
    acrfac = 2.*rslope3(i,k,2)+2.*diameter*rslope2(i,k,2) &
                +diameter**2*rslope(i,k,2)
    psaci(i,k) = pi*qi(i,k,j)*eacrs*n0s*n0sfac(i,k) &
                      *abs(vt2s-vt2i)*acrfac/4.
  endif
  if(qs(i,k,j).gt.qcrmin.and.qc(i,k,j).gt.qmin) then
    psacw(i,k) = min(pacrc*n0sfac(i,k)*rslope3(i,k,2) &
                      *rslopeb(i,k,2)*qc(i,k,j)*denfac(i,k) &
                      ,qc(i,k,j)*rdtcld)
  endif
  if(qi(i,k,j).gt.0.and.ifsat.ne.1) then
    xmi = den(i,k,j)*qi(i,k,j)/xni(i,k)
    diameter = dicon * sqrt(xmi)
    pidep(i,k) = 4.*diameter*xni(i,k)*(rh(i,k,2)-1.)/work1(i,k,2)
    supice = satdt-prevp(i,k)
    if(pidep(i,k).lt.0.) then
      pidep(i,k) = max(max(pidep(i,k),satdt*.5),supice)
      pidep(i,k) = max(pidep(i,k),-qi(i,k,j)*rdtcld)
    else
      pidep(i,k) = min(min(pidep(i,k),satdt*.5),supice)
    endif
    if(abs(prevp(i,k)+pidep(i,k)).ge.abs(satdt)) ifsat = 1
  endif
endif
if(qs(i,k,j).gt.0..and.ifsat.ne.1) then
  coeres = rslope2(i,k,2)*sqrt(rslope(i,k,2)*rslopeb(i,k,2))
  psdep(i,k) = (rh(i,k,2)-1.)*n0sfac(i,k) &
                  *(precs1*rslope2(i,k,2)+precs2 &
                  *work2(i,k)*coeres)/work1(i,k,2)
  supice = satdt-prevp(i,k)-pidep(i,k)
  if(psdep(i,k).lt.0.) then
    psdep(i,k) = max(psdep(i,k),-qs(i,k,j)*rdtcld)
    psdep(i,k) = max(max(psdep(i,k),satdt*.5),supice)
  else
    psdep(i,k) = min(min(psdep(i,k),satdt*.5),supice)
  endif
  if(abs(prevp(i,k)+pidep(i,k)+psdep(i,k)).ge.abs(satdt)) &
    ifsat = 1
endif
if(supcol.gt.0) then
  if(supsat.gt.0.and.ifsat.ne.1) then
    supice = satdt-prevp(i,k)-pidep(i,k)-psdep(i,k)
    xni0 = 1.e3*exp(0.1*supcol)
    roqi0 = 4.92e-11*exp(log(xni0)*(1.33))
    pigen(i,k) = max(0.,(roqi0/den(i,k,j)-max(qi(i,k,j),0.)) &
                    *rdtcld)
    pigen(i,k) = min(min(pigen(i,k),satdt),supice)
  endif
  if(qi(i,k,j).gt.0.) then
    qimax = roqimax/den(i,k,j)
    psaut(i,k) = max(0.,(qi(i,k,j)-qimax)*rdtcld)
  endif
endif
if(supcol.lt.0.) then
  if(qs(i,k,j).gt.0..and.rh(i,k,1).lt.1.) &
    psevp(i,k) = psdep(i,k)*work1(i,k,2)/work1(i,k,1)
  psevp(i,k) = min(max(psevp(i,k),-qs(i,k,j)*rdtcld),0.)
endif
enddo
enddo
do k = kts, kte
  do i = its, ite
    if(t(i,k,j).le.t0c) then
      value = max(qmin,qc(i,k,j))
      source = (praut(i,k)+pracw(i,k)+psacw(i,k))*dtcld
      if (source.gt.value) then
        factor = value/source
        praut(i,k) = praut(i,k)*factor
```

```
      pracw(i,k) = pracw(i,k)*factor
      psacw(i,k) = psacw(i,k)*factor
    endif
    value = max(qmin,qi(i,k,j))
    source = (psaut(i,k)+psaci(i,k)-pigen(i,k)-pidep(i,k))*dtcld
    if (source.gt.value) then
      factor = value/source
      psaut(i,k) = psaut(i,k)*factor
      psaci(i,k) = psaci(i,k)*factor
      pigen(i,k) = pigen(i,k)*factor
      pidep(i,k) = pidep(i,k)*factor
    endif
    work2(i,k)=-(prevp(i,k)+psdep(i,k)+pigen(i,k)+pidep(i,k))
    q(i,k,j) = q(i,k,j)+work2(i,k)*dtcld
    qc(i,k,j) = max(qc(i,k,j)-(praut(i,k)+pracw(i,k) &
                    +psacw(i,k))*dtcld,0.)
    qr(i,k,j) = max(qr(i,k,j)+(praut(i,k)+pracw(i,k) &
                    +prevp(i,k))*dtcld,0.)
    qi(i,k,j) = max(qi(i,k,j)-(psaut(i,k)+psaci(i,k) &
                    -pigen(i,k)-pidep(i,k))*dtcld,0.)
    qs(i,k,j) = max(qs(i,k,j)+(psdep(i,k)+psaut(i,k) &
                    +psaci(i,k)+psacw(i,k))*dtcld,0.)
    xlf = xls-xl(i,k)
    xlwork2 = -xls*(psdep(i,k)+pidep(i,k)+pigen(i,k)) &
                 -xl(i,k)*prevp(i,k)-xlf*psacw(i,k)
    t(i,k,j) = t(i,k,j)-xlwork2/cpm(i,k)*dtcld
  else
    value = max(qmin,qc(i,k,j))
    source=(praut(i,k)+pracw(i,k)+psacw(i,k))*dtcld
    if (source.gt.value) then
      factor = value/source
      praut(i,k) = praut(i,k)*factor
      pracw(i,k) = pracw(i,k)*factor
      psacw(i,k) = psacw(i,k)*factor
    endif
    value = max(qcrmin,qs(i,k,j))
    source=(-psevp(i,k))*dtcld
    if (source.gt.value) then
      factor = value/source
      psevp(i,k) = psevp(i,k)*factor
    endif
    work2(i,k)=-(prevp(i,k)+psevp(i,k))
    q(i,k,j) = q(i,k,j)+work2(i,k)*dtcld
    qc(i,k,j) = max(qc(i,k,j)-(praut(i,k)+pracw(i,k) &
                    +psacw(i,k))*dtcld,0.)
    qr(i,k,j) = max(qr(i,k,j)+(praut(i,k)+pracw(i,k) &
                    +prevp(i,k) +psacw(i,k))*dtcld,0.)
    qs(i,k,j) = max(qs(i,k,j)+psevp(i,k)*dtcld,0.)
    xlf = xls-xl(i,k)
    xlwork2 = -xl(i,k)*(prevp(i,k)+psevp(i,k))
    t(i,k,j) = t(i,k,j)-xlwork2/cpm(i,k)*dtcld
  endif
 enddo
enddo
hsub = xls
hvap = xlv0
cvap = cpv
ttp=t0c+0.01
dldt=cvap-cliq
xa=-dldt/rv
xb=xa+hvap/(rv*ttp)
dldti=cvap-cice
xai=-dldti/rv
xbi=xai+hsub/(rv*ttp)
do k = kts, kte
  do i = its, ite
    tr=ttp/t(i,k,j)
    lqs(i,k,1)=psat*exp(log(tr)*(xa))*exp(xb*(1.-tr))
    lqs(i,k,1) = ep2 * lqs(i,k,1) / (p(i,k,j) - lqs(i,k,1))
    lqs(i,k,1) = max(lqs(i,k,1),qmin)
    tr=ttp/t(i,k,j)
    if(t(i,k,j).lt.ttp) then
      lqs(i,k,2)=psat*exp(log(tr)*(xai))*exp(xbi*(1.-tr))
    else
```

```
        lqs(i,k,2)=psat*exp(log(tr)*(xa))*exp(xb*(1.-tr))
      endif
      lqs(i,k,2) = ep2 * lqs(i,k,2) / (p(i,k,j) - lqs(i,k,2))
      lqs(i,k,2) = max(lqs(i,k,2),qmin)
    enddo
  enddo
  do k = kts, kte
    do i = its, ite
      work1(i,k,1) = ((max(q(i,k,j),qmin)-(lqs(i,k,1)))/ &
          (1.+(xl(i,k))*(xl(i,k))/(rv*(cpm(i,k)))*(lqs(i,k,1))/&
                              ((t(i,k,j))*(t(i,k,j)))))
      work2(i,k) = qc(i,k,j)+work1(i,k,1)
      pcond(i,k) = min(max(work1(i,k,1)/dtcld,0.),max(q(i,k,j),0.)/dtcld)
      if(qc(i,k,j).gt.0..and.work1(i,k,1).lt.0.) &
            pcond(i,k) = max(work1(i,k,1),-qc(i,k,j))/dtcld
      q(i,k,j) = q(i,k,j)-pcond(i,k)*dtcld
      qc(i,k,j) = max(qc(i,k,j)+pcond(i,k)*dtcld,0.)
      t(i,k,j) = t(i,k,j)+pcond(i,k)*xl(i,k)/cpm(i,k)*dtcld
    enddo
  enddo
  do k = kts, kte
    do i = its, ite
      if(qc(i,k,j).le.qmin) qc(i,k,j) = 0.0
      if(qi(i,k,j).le.qmin) qi(i,k,j) = 0.0
    enddo
  enddo
ENDDO
!$acc end region
```

```
DO j=jts,jte
  pi = 4. * atan(1.)
  do k = kts, kte
    do i = its, ite
      qc(i,k,j) = max(qc(i,k,j),0.0)
      qr(i,k,j) = max(qr(i,k,j),0.0)
      qi(i,k,j) = max(qi(i,k,j),0.0)
      qs(i,k,j) = max(qs(i,k,j),0.0)
    enddo
  enddo
```

```
__attribute__((__global__))
pgi_kernel_2(
  __attribute__((__shared__)) int i1,
  __attribute__((__shared__)) int tc3,
  __attribute__((__shared__)) int tc2,
  __attribute__((__shared__)) float* _qc,
  __attribute__((__shared__)) int uqc_1,
  __attribute__((__shared__)) int uqc_2,
  __attribute__((__shared__)) int _jts,
  __attribute__((__shared__)) int _kts,
  __attribute__((__shared__)) int _its,
  __attribute__((__shared__)) float* _qr,
  __attribute__((__shared__)) int uqr_1,
  __attribute__((__shared__)) int uqr_2,
  __attribute__((__shared__)) float* _qi,
  __attribute__((__shared__)) int uqi_1,
  __attribute__((__shared__)) int uqi_2,
  __attribute__((__shared__)) float* _qs,
  __attribute__((__shared__)) int uqs_1,
  __attribute__((__shared__)) int uqs_2 )
{
  int i3, i2;
 /* dogang i2  (loop:58 ploop:2) */
  i2 = blockIdx.x;
    /* dogang i3  (loop:59 ploop:3) */
      /* dosimd i3  (loop:59 ploop:4) */
      i3 = threadIdx.x + blockDim.x*blockIdx.y;
      _qc[i3+uqc_1*(i2+uqc_2*(i1))] =
        max(_qc[i3+uqc_1*(i2+uqc_2*(i1))],0.00000000e+00);
      _qr[i3+uqr_1*(i2+uqr_2*(i1))] =
        max(_qr[i3+uqr_1*(i2+uqr_2*(i1))],0.00000000e+00);
      _qi[i3+uqi_1*(i2+uqi_2*(i1))] =
        max(_qi[i3+uqi_1*(i2+uqi_2*(i1))],0.00000000e+00);
      _qs[i3+uqs_1*(i2+uqs_2*(i1))] =
        max(_qs[i3+uqs_1*(i2+uqs_2*(i1))],0.00000000e+00);

}
```

```
DO j=jts,jte
  do k = kts, kte
    do i = its, ite
      cpm(i,k) = cpmcal(q(i,k,j))
      xl(i,k) = xlcal(t(i,k,j))
    enddo
  enddo
```

```
__attrbute__((__global__))
pgi_kernel_5(
  __attribute__((__shared__)) int i1,
  __attribute__((__shared__)) int tc3,
  __attribute__((__shared__)) int tc2,
  __attribute__((__shared__)) float _cpv,
  __attribute__((__shared__)) float _qmin,
  __attribute__((__shared__)) float* _q,
  __attribute__((__shared__)) int uq_1,
  __attribute__((__shared__)) int uq_2,
  __attribute__((__shared__)) int _jts,
  __attribute__((__shared__)) int _kts,
  __attribute__((__shared__)) int _its,
  __attribute__((__shared__)) float _cpd,
  __attribute__((__shared__)) float* _cpm,
  __attribute__((__shared__)) int ucpm_1,
  __attribute__((__shared__)) float _xlv0,
  __attribute__((__shared__)) float _xlv1,
  __attribute__((__shared__)) float* _t,
  __attribute__((__shared__)) int ut_1,
  __attribute__((__shared__)) int ut_2,
  __attribute__((__shared__)) float _t0c,
  __attribute__((__shared__)) float* _xl,
  __attribute__((__shared__)) int uxl_1 )
{
  int i3, i2;
  /* dogang i2  (loop:56 ploop:5) */
  i2 = blockIdx.x;
    /* dogang i3  (loop:57 ploop:6) */
      /* dosimd i3  (loop:57 ploop:7) */
      i3 = threadIdx.x + blockDim.x*blockIdx.y;
      _cpm[i3+ucpm_1*(i2)] =
         ((_cpv*max(_qmin,_q[i3+uq_1*(i2+uq_2*(i1))]))+
          (_cpd*(1.00000000e+00-max(_qmin,_q[i3+uq_1*(i2+uq_2*(i1))]))));
      _xl[i3+uxl_1*(i2)] = (_xlv0-(_xlv1*(_t[i3+ut_1*(i2+ut_2*(i1))]-_t0c)));
}
```

The Portland Group

```
wsm5:
  271, Loop is fully parallel
       Parallelization requires privatization of array work2c(its:ite,kts:kte)
       Parallelization requires privatization of array work1c(its:ite,kts:kte)
       Parallelization requires privatization of array work2(its:ite,kts:kte)
       Parallelization requires privatization of array rmstep(its:ite)
       Parallelization requires privatization of array work1(its:ite,kts:kte,1:2)
       Parallelization requires privatization of array numdt(its:ite)
       Parallelization requires privatization of array rslope3(its:ite,kts:kte,1:2)
       Parallelization requires privatization of array rslope2(its:ite,kts:kte,1:2)
       Parallelization requires privatization of array rslopeb(its:ite,kts:kte,1:2)
       Parallelization requires privatization of array rslope(its:ite,kts:kte,1:2)
       Parallelization requires privatization of array n0sfac(its:ite,kts:kte)
       Parallelization requires privatization of array xni(its:ite,kts:kte)
       Parallelization requires privatization of array falkc(its:ite,kts:kte)
       Parallelization requires privatization of array fallc(its:ite,kts:kte)
       Parallelization requires privatization of array fall(its:ite,kts:kte,1:2)
       Parallelization requires privatization of array falk(its:ite,kts:kte,1:2)
       Parallelization requires privatization of array psevp(its:ite,kts:kte)
       Parallelization requires privatization of array psmlt(its:ite,kts:kte)
       Parallelization requires privatization of array pcond(its:ite,kts:kte)
       Parallelization requires privatization of array pidep(its:ite,kts:kte)
       Parallelization requires privatization of array pigen(its:ite,kts:kte)
       Parallelization requires privatization of array psacw(its:ite,kts:kte)
       Parallelization requires privatization of array psaci(its:ite,kts:kte)
       Parallelization requires privatization of array pracw(its:ite,kts:kte)
       Parallelization requires privatization of array psaut(its:ite,kts:kte)
       Parallelization requires privatization of array praut(its:ite,kts:kte)
       Parallelization requires privatization of array psdep(its:ite,kts:kte)
       Parallelization requires privatization of array prevp(its:ite,kts:kte)
       Parallelization requires privatization of array rh(its:ite,kts:kte,1:2)
       Parallelization requires privatization of array lqs(its:ite,kts:kte,1:2)
       Parallelization requires privatization of array denfac(its:ite,kts:kte)
       Parallelization requires privatization of array mstep(its:ite)
       Parallelization requires privatization of array xl(its:ite,kts:kte)
       Parallelization requires privatization of array cpm(its:ite,kts:kte)
```

PGI Accelerator Schedule

- Define directive-based programming model
- Create production-quality Accelerator compiler
- PGI Compilers will detect and report Compute Intensity
- PGPROF upgrade to assist in tuning for Accelerators
- Investigating non-CUDA targets
- PGI will demo Accelerator Compilers at SC|08

PGI Premier Conclusions

- Delivers education to better use compilers and tools
- Provides direct scientist to engineer interaction
- Provides custom compiler and library work
- Provides a compiler engineer for your code development team
- Results in faster application results!

The Portland Group

Additional Material

The Portland Group

What's New in PGI 8.0

- OpenMP 3.0 Support
- Continued SPECFP06 and SPECINT06 Performance
- PGPROF improvements
- PGI Unified Binary enhancements
- Common Compiler Feedback Format
- Tuning for AMD Shanghai processors
- Accelerator Compiler Beta
- Improved C++ STL performance, features
- Bug fixes

10 Steps to Multi-core Performance

- Use correct target processor, -tp barcelona-64
- Vectorization and single core performance is a good start
- The PGI -Mconcur flag can handle simple cases, and might surprise you with where it can find parallelism
- OpenMP gives finer control, is supported everywhere
- Don’t fret. Gather some profiling data on where cache misses or other delays occur

10 Steps to Multi-core Performance

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

Compute Intensity of Size N = 1024 Complex-to-Complex Radix-2 FFT (SP)

$$\mathbf{I} = \frac{5N\text{Log}_2(N)}{4N} = \frac{5\text{Log}_2(N)}{4} = 12.5 \text{ FLOPS / WORD}$$

$$12.5 \frac{\text{FLOPS}}{\text{Word}} \times \frac{0.5 \text{ GWORDS}}{\text{Second}} = 6.25 \text{ GFLOPS}$$

Sustained Memory-to-Memory Performance Potential on a 16x PCI express slot, assuming 4Gbytes/sec one-way

Compute Intensity of DP MATMUL

$$\mathbf{I} = \frac{2N}{3} = \begin{cases} 6.7 \text{ FLOPS / Word at } 10\text{x}10 \\ 66.7 \text{ FLOPS / Word at } 100\text{x}100 \\ 667.0 \text{ FLOPS / Word at } 1000\text{x}1000 \end{cases}$$

$$667 \frac{\text{FLOPS}}{\text{Word}} \times \frac{0.25 \text{ GWords}}{\text{Second}} = 167 \text{ GFLOPS}$$

Sustained Memory-to-Memory Performance Potential
on a 16x PCI express slot, assuming 4Gbytes/sec one-way

The Portland Group

PGI Products Unique Features

- Outstanding 64-bit performance on *both* Intel 64 (Core 2) and AMD Opteron – fully-tuned for Barcelona
- PGI Unified Binary technology
- Complete/Integrated HPC developer suite - compile/debug/profile
- MPI debugging & profiling for Windows CCS Clusters
- Completely uniform cross-platform development for multi-core AMD, Intel, Linux, Windows, SUA, MacOS
- Self-contained MPI/OpenMP development for Multi-core systems and clusters
- Fully-supported on Windows SUA (Interix)
- Comprehensive UNIX => x64 migration options

Using the -Mconcur Option

`-Mconcur[=`*`option`*`[,`*`option`*`]]` where *`option`* is:

`[no]altcode:<n>`	[Don't] Generate alternate scalar code for parallel loops
`dist:block`	Parallelize with block distribution (default)
`dist:cyclic`	Parallelize with cyclic distribution
`cncall`	Loops with calls are candidates for parallelization
`noassoc`	Disable parallelization of loops with reductions
`innermost`	Enable parallelization of innermost loops
`levels:<n>`	Parallelize loops nested at most **n** levels deep
`[no]numa`	[Don't] Use thread affinity for NUMA architectures