

PGI[®] Compilers and Tools for AMD Opteron Processor-based CRAY Systems

*Cray User Group Meeting
May 2005*

Doug Miles – douglas.miles@st.com

STMicroelectronics
The Portland Group
www.pgroup.com



Outline

- **CRAY/PGI Relationship**
- **PGI Compiler Architecture and Features**
- **Applications Porting/Tuning Resources**
- **PGI Compilers Basic Usage, Important Options**
- **Vectors are Back!**
- **Pending Tuning Issues, PGI Compilers Roadmap**



CRAY + PGI Relationship

- **CRAY/PGI History** – PGHPP/CRAFT for CRAY T3E (1996)
- **PGI ASCI History** – ASCI Red F90/C/C++ for Sandia (1997)
- **PGI on Linux86** – Production F90/C/C++ compilers (1997)
- **PGI on AMD Opteron** – Cooperation with AMD (2002)
- **ASCI + CRAY + Opteron Red Storm** → CRAY + PGI
- **CRAY/PGI Agreement** – CRAY resells and supports PGI F95/C/C++ on Red Storm, CRAY XT3, CRAY XD1 (2004)

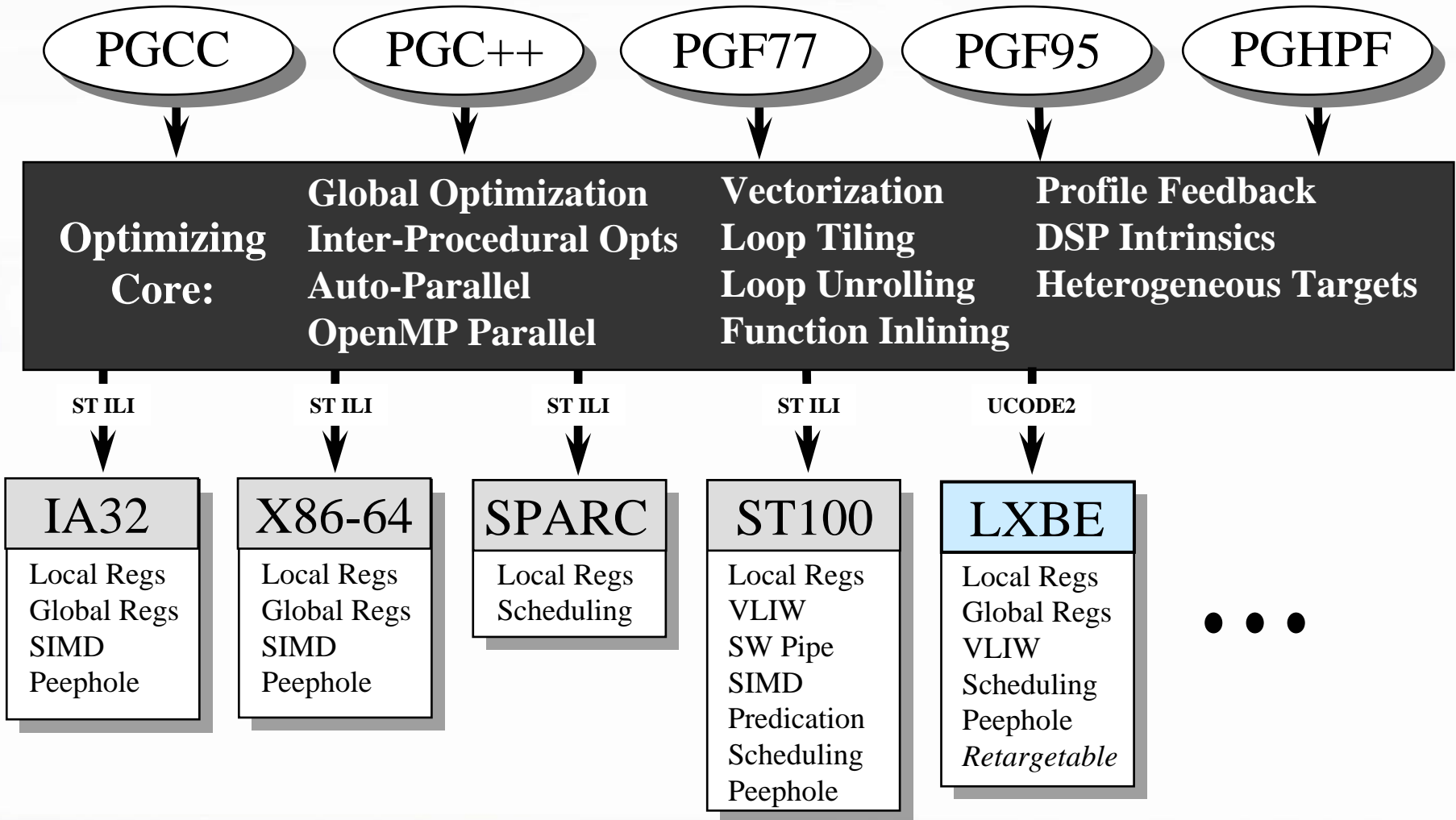


Who Does What?

- **Development** – Compiler development/QA by PGI
- **CRAY Integration** – Add'l CRAY testing and integration on target HW and OS
- **Applications, Benchmarking** – Both
- **Technical Support** – Frontline support by CRAY, compiler bugs to PGI for fixing and (if needed) help finding bug workarounds
- **Release Schedules** – Standard PGI release schedules; usually 2 per year (one major, one minor) and several “builds” of each release for tech support / new OSs / etc; current release is **6.0-4**



PGI Compilers



PGI Compilers Key Features

- **Optimization** – State-of-the-art optimization infrastructure
- **Cross-platform** – AMD & Intel, 32- & 64-bit, Linux & Windows
- **Tools** – F95/C/C++ debugger/profiler, doc's, pgroup.com
- **Comprehensive Linux Support** – Red Hat 7.3 – 9.0, RHEL 3.0/4.0, Fedora Core 2/3, SuSE 7.1 – 9.2, SLES 8/9
- **Parallel** – OpenMP/MPI supported in all languages and tools
- **Infrastructure** – NAG, VNI, ACML, TotalView, ISV App's, Research App's, Research Tools, PAPI



Commercial Applications Porting to 64-bit x86 with PGI Compilers

- **MCAE** – ANSYS*, ADINA*, MSC.MARC*, NX NASTRAN
- **Computational Chemistry** – GAUSSIAN*
- **Automotive** – LS-DYNA*, PAM-CRASH, PAM-STAMP, RADIOSS*, MADYMO
- **CFD** – STAR-CD*, Fluent POLYFLOW*, AVL Fire*
- **Geophysical** – Several Proprietary
- **Math Libraries** – ACML*, NAG*

*In production



Research Applications Pre-tested with PGI Compilers

- **Weather** – MM5, WRF2, POP, MOM4, CAM
- **Computational Chemistry** – GAMESS, AMBER, MOLPRO, CHARMM, PWSCF
- **Bioinformatics** – BLAST
- **DOE/DOD** – MCNP5, TBMD
- **Libraries** – ATLAS, OPENGL, NetCDF, MPICH, MPICH-GM





pgroup.com

- License/Seat Mgmt
- Documentation
- Online FAQs
- Online User Forums
- Extensive App's porting and tuning guides



The Portland Group

PGI | Resources | 64-bit GAMESS Tips & Techniques - Netscape

New Tab | PGI | Resources | 64-bit GAMESS Tips & Te...

Version Information

This guide was created for the GAMESS November 22, 2004 version and PGI Release 6.0 pgf90 and pgcc 64-bit compilers on a 64-bit Linux system. Both AMD™ AMD64/Opteron and Intel® Xeon with EM64T are supported.

Application Notes

Information about GAMESS can be found at the [GAMESS home page](#). GAMESS is maintained by the members of the Gordon research group at Iowa State University. From the GAMESS Home page:

"GAMESS is a program for ab initio quantum chemistry. Briefly, GAMESS can compute SCF wave functions ranging from RHF, ROHF, UHF, GVB, and MCSCF. Correlation corrections to these SCF wave functions include Configuration Interaction, second order perturbation theory, and Coupled-Cluster approaches, as well as the Density Functional Theory approximation. Analytic gradients are available, for automatic geometry optimization, transition state searches, or reaction path following. Computation of the energy hessian permits prediction of vibrational frequencies. The chart below summarizes the program's present capabilities for obtaining wave functions, applying correlation treatments, and computing derivatives. A variety of molecular properties, ranging from simple dipole moments to frequency dependent hyperpolarizabilities may be computed. Many basis sets are stored internally, and together with effective core potentials, all elements up to Radon may be included in molecules. Several graphics programs are available for viewing of the final results. Many of the computational functions can be performed using direct techniques, or in parallel on appropriate hardware."

Obtaining the Source Code

Information on obtaining the GAMESS source code can be found at the [US Government's Ames Laboratory](#). The Ames lab requires you to register before downloading the source code.

Dependencies

No known dependencies.

Configuration and Set-up Information

Once unpacked, please read the enclosed documentation (*.DOC) files and the "readme.unix" file in the "misc" directory. To build and run GAMESS, you will first need to edit several scripts (ddi/compddi, compall, comp, lkcd, runngms, runall). Due to license restrictions on the source code, we cannot make available pre configured versions of these files. Note that the line numbers may change with each release of GAMESS.

ddi/compddi	<ol style="list-style-type: none"> 1. Line 17: set the "TARGET" to "amd64".
compall	<ol style="list-style-type: none"> 1. Line 15: set the "TARGET" to "amd64". 2. Line 16: replace the "/u1/mike/gameSS" with the root directory of your GAMESS package. 3. Line 36: Change the CCOMP flags for amd64 to: <pre>if (\$TARGET == amd64) set CCOMP='pgcc -fastsse'</pre>
comp	<ol style="list-style-type: none"> 1. Line 17: set the "TARGET" to "amd64". 2. Line 18: replace the "/u1/mike/gameSS" with the root directory of your GAMESS package. 3. Line 47: If you wish to use a BLAS library, i.e. "-lblas" or "-lacml", set "BLAS3=true". 4. On line 533 set the OPT to <pre>set OPT = '-fastsse'</pre> 5. On line 536, remove "-Mprof=func" from OPT. 6. On line 538, you can use either pgf77 or pgf90. Add "-Mfixed" if you prefer pgf90.
lkcd	<ol style="list-style-type: none"> 1. Line 17: set the "TARGET" to "amd64".

Done

GAMESS Application Note on pgroup.com



Basic Usage of PGI Compilers

- **Compiler *drivers*** – interpret options and invoke pre-processing, compilers, IPA, assembler, linker
- **Options precedence** – if two or more options conflict, last option on command line takes precedence
- **Use `-help`** – to list all options or see details on how to use a given option, e.g. `pgf95 -fastsse -help`
- **Use `-minfo`** – to see a compile-time listing of optimizations and transformations performed by the compiler
- **PGI User's Guide** – chapters 2 and 3 – 35 pages that will make you an expert at using the PGI compilers



Important PGI Compiler Options

- fast** Includes “-O2 -Munroll -Mnoframe -Mlre”
- fastsse** Includes “-fast -Mvect=sse -Mcache_align”
- 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
- help** Compiler options and usage

-fastsse -Mipa=fast usually best for “compile-and-go”



Vectorizable Loop in SPEC FP2K 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 use 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(%eax),%xmm1
movss -8(%eax),%xmm2
movss -4(%eax),%xmm3
decl %edx
mulss -12(%ecx),%xmm1
addss -572(%ebp),%xmm1
mulss -8(%ecx),%xmm2
addss %xmm2,%xmm1
mulss -4(%ecx),%xmm3
addss %xmm3,%xmm1
movss (%eax),%xmm2
addl $16,%eax
mulss (%ecx),%xmm2
addl $16,%ecx
addss %xmm2,%xmm1
testl %edx,%edx
movss %xmm1,-572(%ebp)
jg .LB6_668
```

Vector SSE:

.LB6_1105:

lineno: 358

```
movlps (%esi,%ecx),%xmm2
movlps (%edx,%ecx),%xmm3
movhps 8(%esi,%ecx),%xmm2
movhps 8(%edx,%ecx),%xmm3
mulps %xmm2,%xmm3
movlps 16(%esi,%ecx),%xmm2
movhps 24(%esi,%ecx),%xmm2
addps %xmm3,%xmm0
movlps 16(%edx,%ecx),%xmm3
movhps 24(%edx,%ecx),%xmm3
addl $32,%ecx
mulps %xmm2,%xmm3
addps %xmm3,%xmm0
subl $8,%eax
jg .LB6_1105
```

Facerec Scalar: 104.2 sec
Facerec Vector: 84.3 sec



Maximizing Vectorization

- May need to split “large” loops by hand (or try `-Mvect=nosizelimit`)
- Don't unroll loops by hand – re-roll if necessary to enable vectorization and let the compiler unroll loops where needed (or use `-Munroll`)
- Fortran 90 POINTER may not be contiguous
- Index arrays can prevent vectorization, or make it inefficient
- Check asm code to see if `movaps` / `movapd` used
- Vectorization is not always profitable, especially double-precision
- Double-precision scalar vs vector peak speed is the same, but vectorized loops have fewer instructions, can use aligned moves



Pending Compiler Tuning Issues

- **C/C++ Inlining** – funcs returning struct, funcs w/statics, alloca, operators
- **C/C++ General** – global optz across non-loop blocks, scalar replacement of aggregates, better idiom recognition, mitigating load after store penalties (primarily for EM64T), eliminate useless struct copies
- **C++** – re-implement trap handler / --no_exceptions, EDG IL optz
- **OpenMP PARALLEL DO iterations** – alignment versus load balancing
- **OpenMP thread CPU hopping** – sched_setaffinity() a little flakey
- **NUMA on AMD Opteron** – ensure memory is node-local, need NUMA-aware OpenMP runtime library
- **Autoparallel on Dual-core** – How effective can we make it?
- **Prefetching** – non-vector loops, prefetching for indirect, directives
- **EM64T-specific Issues** – seem to be a lot of them; characterizing



PGI Compilers & Tools Roadmap

- **PGI 6.0-5 in June, PGI 6.1 in November**
- **More Performance Tuning**
 - C/C++ tuning – IPA, inlining, pointer opt's, EDG IL opt's, etc (6.1)
 - Dual-core – Support/Tuning/OpenMP (6.0-5, more in 6.1)
 - OpenMP – DYNAMIC, GUIDED, runtime efficiency (6.0-5), NUMA (6.1)
 - AMD64, IA32, EM64T code generator tuning (6.1)
 - F95 optimizations (6.1 – need examples of object-oriented F95!)
 - Vectorizer – splitting, partial vect, transcendentals, conditionals, etc (6.1)
 - Track/tune more end-user & ISV applications (ongoing)
 - Hand-coded math libs (6.1)
- **Language/Compiler Features** – C99, OpenMP 2.5, SSE3, GNU asm (6.1)
- **Tools Features** – PGI Visual Fortran; VS05 integration on Win64
- **Documentation** – Can we simplify?
- **Operating Systems** – SuSE, Red Hat, 64-bit Windows



Questions?

