Center for High Performance Software Research

Overview

Ken Kennedy
Rice University

http://www.cs.rice.edu/~ken/Presentations/HiPerSoftIBM06.pdf
HiPerSoft Themes

• Traditional High-Performance Technical Computing
  — Parallelism
  — Memory hierarchy
  — Performance analysis and prediction
  — Multicore computing, including Cell
  — Autotuning
  — Compiler support for modern languages (Java, Matlab, HPF, HPCS)

• Grid Computing
  — Heterogeneous platforms
  — Generic scheduling based on performance models
  — High level specification of workflows

• Telescoping Languages
  — High performance domain-specific systems based on scripting languages
  — Precompilation of domain libraries
HiPerSoft

- Los Alamos Computer Science Institute (LACSI)
  - LANL, Rice, UH, Tennessee, UNC, UNM
  - Enhance productivity on scalable parallel supercomputers

- VGrADS ITR
  - Software support for Grid application development

- Telescoping Languages Project
  - Domain languages based on Matlab, R, Python, LabView
  - Funding from NSF, DOE ASC, DOD, National Instruments

- DOE Office of Science Projects
  - Co-Array Fortran, HPCS languages, performance evaluation
  - Proposed SciDAC Institute, including open source development

- Center for Research on Multicore Computing
  - Funding from Microsoft, Intel (both pending)

- Gulf Coast Center For Computational Cancer Research

- Participants in two MRI (Major Research Infrastructure) grants
Senior Researchers

Ken Kennedy, Director
John Mellor-Crummey, Deputy Director
Rob Fowler, Associate Director
Linda Torczon, Executive Director
Zoran Budimlic
Keith Cooper
Tim Harvey
Guohua Jin
Charles Koelbel

Plus: Rice recruiting to open HPC faculty slot
Philosophy

• Productivity is critical
• Performance is critical
• We can have both

How we work

— Application collaborations
  - Understand impediments to productivity and performance
  - Improvements implemented by hand
— Long-term compiler and software research
  - Focus on the ultimate goal
— Short to medium-term software byproducts
  - Spin out useful tools
  - Mid-term course corrections
Research Strategy

Hard Problem

Solution Vision

Usable Tool

Ultimate Goal

Present

Near Future

Far Future
Scalable Application Development

- Scalable from high-level to high-performance
- Scalable from uniprocessor (or multicore) to high-end parallel
- Scalable from single platform to multiple platforms

---

Center for High Performance Software Research
Research I

• Performance analysis and anomaly detection
  — HPCToolkit, memory analysis, automated parallel anomaly detection
    - All based on binary analysis and sample-based profiling
  — Cross-platform performance prediction

• Languages and compilers for parallel computing
  — HPF, Co-Array Fortran, HPCS languages
  — Optimization of object-oriented languages
    - Object inlining, type analysis, fixing dynamic dispatch

• Compilation for new architectures
  — Multicore
  — Vector units on chip (SSE, Altivec, Cell)
  — heterogeneous processors (grid in a box or on a chip)
Research II

• High-level domain languages
  – Telescoping languages
    – Pre-optimization of libraries
  – Scripting languages:
    – Matlab, R, Python, LabView,
    – Fortran, C++
  – Parallelism in scripting languages
    – Parallel Matlab based on HPF compiler infrastructure

• High performance component integration
  – Another application of telescoping languages
    – Trade off some flexibility for performance

• Automatic tuning for new platforms
  – Automatic search space generation
  – Tuning for new platforms based on trial runs
Why Rice?

• **Experience**
  - High end compilers and tools
  - Leadership in managing community efforts (CRPC, LACSI, VGrADS)

• **Research Base**
  - Performance analysis tools
    - HPCToolkit and performance prediction infrastructure
  - High-level languages
    - HPF, Co-Array Fortran, HP Java, Parallel Matlab
  - Component integration and domain languages
    - Telescoping languages framework
    - Matlab, R, Python, LabView
  - Heterogeneous architectures
    - Grid application development software

• **Open-source Software**
  - Open64, D System, Telescoping Languages, VGrADS, JaMake
  - ROSE as a collaboration with LLNL
Multicore and Cell

Compilers and Tools for Next-Generation Chip Architectures
Bandwidth Management

• Multicore raises computational power rapidly
  — Bandwidth onto chip unlikely to keep up

• Multicore systems will feature shared caches
  — Replaces false sharing with enhanced probability of conflict misses

• Challenges for effective use of bandwidth
  — Enhancing reuse when multiple processors are using cache
  — Reorganizing data to increase density of cache block use
  — Reorganizing computation to ensure reuse of data by multiple cores
    - Inter-core pipelining
  — Managing conflict misses
    - With and without architectural help

• Without architectural help
  — Data reorganization within pages and synchronization to minimize conflict misses
    - May require special memory allocation run-time primitives
Conflict Misses

• Unfortunate fact:
  - If a scientific calculation is sweeping across strips of > k arrays on a machine with k-way associativity and
  - All k strips overlap in one associativity group, then
    - Every access to the overlap group location is a miss

On each outer loop iteration, 1 evicts 2 which evicts 3 which evicts 1
In a 2-way associative cache, all are misses!

This limits loop fusion, a profitable reuse strategy
Controlling Conflicts: An Example

• **Cache and Page Parameters**
  - 256K Cache, 4-way set associative, 32-byte blocks
    - 1024 associativity groups
  - 64K Page
    - 2048 cache blocks
  - Each block in a page maps to a unique associativity group
    - 2 different lines in a page map to the same associativity group

• **In General**
  - Let $A$ = number of associativity groups in cache
  - Let $P$ = number of cache blocks in a page
  - If $P \geq A$ then each block in a page maps to a single associativity group
    - No matter where the page is loaded
  - If $P < A$ then a block can map to $A/P$ different associativity groups
    - Depending on where the page is loaded
Questions

• Can we do data allocation precisely within a page so that conflict misses are minimized in a given computation?
  — Extensive work on minimizing self-conflict misses
  — Little work on inter-array conflict minimization
  — No work, to my knowledge, on interprocessor conflict minimization

• Can we synchronize computations so that multiple cores do not interfere with one another?
  — Even reuse blocks across processors

• Might it be possible to convince vendors to provide additional features to help control cache, particularly conflict misses
  — Allocation of part of cache as a scratchpad
  — Dynamic modification of cache mapping
Compiling for Cell

- 1 PPE (AltiVec), 8 SPE (vector) each with 256KB LS
- Peak performance: At 3.2GHz, ~200GFlops for SP, ~20GFlops for DP
- Each SPE has its own address space and can only access data from LS, data transfer explicitly controlled through DMA
  - Each SPE can transfer 8 bytes/cycle each direction
  - total memory bandwidth 25.6 GB/s = 8 bytes per cycle total
Compiler Challenges

• Implications of Bandwidth versus Flops
  — If a computation needs one input word per flop, SPEs must wait for data
    - Eight SPEs can each get 16 bytes (4 words) every 16 cycles
    - Peak under those conditions = 8 flops per 16 cycles = 1.6 Gflops
  — Reuse in LS is essential to get closer to peak

• Compiler Challenges
  — Find sufficient parallelism for SPE
    - coarse and fine granularity
  — Effective data movement strategy
    - DMA latency hiding
    - Data reuse in LS
Our Work on Short Vector Units

- **Altivec on PowerPC, SSE on x86**
  - Short vector length (16 bytes), contiguous memory access for vectors, vector intrinsics only available in C/C++
  - Data alignment constraints

<table>
<thead>
<tr>
<th>Problem size</th>
<th>800</th>
<th>1600</th>
<th>3200</th>
<th>6400</th>
</tr>
</thead>
<tbody>
<tr>
<td>load_ps</td>
<td>0.75</td>
<td>1.42</td>
<td>3.21</td>
<td>6.37</td>
</tr>
<tr>
<td>loadu_ps</td>
<td>1.12</td>
<td>2.21</td>
<td>4.69</td>
<td>9.38</td>
</tr>
</tbody>
</table>
Compilation Strategy for F90

- Procedure outlining array statements and vectorizable loops
  - Rewriting computation in C/C++ to utilize the intrinsics
- Loop alignment to adjust relative data alignment between lhs and rhs
  - In complement to software pipelined vector accesses
- Data padding for data alignment
- Vectorized scalar replacement for data reuse in vector registers
DO J = 2, N-1

\[ A(2:N-1, J) = \frac{A(2:N-1, J-1) + A(2:N-1, J+1) + A(1:N-2, J) + A(3:N, J)}{4} \]

ENDDO

Our Work on Short Vector Units.

- Speedup of cLalign + VPAR to cOrig + V on Pentium IV (Intel Compiler)
- Speedup of cLalign + VPARS to Orig + V on PowerPC G5 (VAST)

Center for High Performance Software Research
Compiling for Computational Accelerators

- Employ integrated techniques for vectorization, padding, alignment, scalar replacement to compile for short vector machines

DO J = 2, N-1;  
ENDDO

1.72  
2.34  
2.39  
2.19

Speedup of  
cLalign+VPAR  
to cOrig+V on  
Pentium IV (Intel Compiler)


- Extending source-to-source vectorizer to support CELL

1.81  
1.88  
1.85  
1.23

Speedup of  
cLalign+VPARS  
to Orig+V on  
PowerPC G5 (VAST)
Related Work on Data Movement

- **Related work: software cache**
  - Eichenberger et al, IBM compiler, cache lookup code before each memory reference, 64KB, 4-way, 128bytes cache line

- **Related work: data management in generated code**
  - Eichenberger et al, loop blocking, prefetching

- **Related work: array copying**
  - Copy regions of a big array into contiguous area of a small array
    - Comparable to DMA transfer
  - Lam et al, reduce cache conflict misses in blocked loop nests
  - Temam et al, cost-benefit models for array copying
  - Yi, general algorithm with heuristics for cost-benefits analysis

- **Related work: software prefetching**
  - Prefetch data into cache before use
    - Comparable to multi-buffering for overlapping computation and communication
  - Callahan et al, Mowry et al, prefetch analysis, prefetch placement
Proposed Work

- Code generation: extending our compiling strategy for attached short vector units (SSE and Altivec) onto the CELL processor
  - Programming model (current)
    - Input: a sequential version Fortran 90 program
    - Output: PPE code + SPE code
    - SPE code obtained by procedure outlining parallelizable loops and array statements
  - Automatic parallelization and vectorization
    - Dependence analysis
    - Multiple SPEs: multiple threads with data/task partitioning
    - Each SPE: vectorization using intrinsics, apply data alignment
Proposed Work.

• **Performance issues**
  - Data alignment at vector and DMA accesses
    - Vector: loop peeling, software pipelined memory accesses, loop alignment, array padding
    - DMA: loop peeling, array padding
  - Data movement
    - Buffer management for multi-buffering
      Buffer size (DMA transfer size)
      Buffer dimensions (DMA transfer placement)
      Number of buffers
    - Data reuse
      Increase the amount of computation per transferred data
      Loop blocking, loop fusion
  - Parallelization enabling transformations
    - Loop distribution, loop interchange, loop skewing, privatization
DMA Size and Computation Per Data

- **Code for sequential version code**

```c
for(i = 0; i < N-2; i++) {
    B[i+1] = 0.0;
    for(k = 0; k < ComputationPerData; k++)
        B[i+1] += (A[i] + A[i+2] + C[i+2]) * (0.35/ComputationPerData);
}
```

- **Parallel version**
  - Range [0 : N-3] split across SPEs
  - Multi-buffering

- **Experiment setup**
  - 2.1GHz CELL blade at UTK
  - Xlc compiler 1.0-2
  - N=22000111, double-buffering
DMA Size and Computation Per Data..

- DMA transfer (floats): 4, 8, 16, 32, 64, 128, 256, 512, 1024, 2048
- Computation Replication: 1, 2, 4, 8, 16, 32, 64, 128, 256