Strategies for High-Performance Heterogeneous Applications: A Compiler Perspective

Mary Hall

October 18, 2006
**Q1: We haven’t solved the homogeneous problem – why work on heterogeneous?**

- Productive parallel programming is still an open problem!
  - Historical commercial focus on driving up clock rates to improve sequential performance
  - Limited funding, usually attached to hardware or application programs
  - Software tools take time to mature

- **Consequence**
  - Parallel programming relegated to small group of experts
  - Fragmented tools community

- **Absence of a migration path for software tools has meant that many innovations in software haven’t made it into practice!**
• High-end (and commodity and embedded) systems are increasingly heterogeneous
  - Roadrunner
  - GPUs in commodity PCs
  - FPGA accelerators (e.g., Cray XD1)
  - Heterogeneous chip architectures such as IBM Cell and Xilinx Virtex 4
    - Domain accelerators such as Clearspeed
• Additional complexity
  - Partitioning of application across functional units
  - Managing data movement and synchronization among processing units
  - Differences in programming models (GP+FPGA is extreme example)
  - Facilitating code reuse
Q3: How will we compile for this new level of complexity?

- Restructure optimizing compilers into modular, understandable chunks
  - Easier to bring up on new platforms
  - Facilitates collaboration

- A systematic, principled approach to application mapping and performance tuning
  - Use vast resources of Petascale systems to improve code performance
  - Enumerate options, try some of them, measure them, keep track of what happened.
Compiler Strategy

Programming Model (Fortran and C for us)

Examples:
- Loop transforms
- Compiler-controlled caching
- Data layout

Key Issues:
- Systematic formulation of optimization search space
- Efficient search techniques
- Feedback from the backends

Target Specific Optimization

FPGA
- Virtex 4
- Cray XD1

GPP+SIMD
- DIVA
- GPUs
- Multimedia
- Cell

TCC
- BG/L

BE

Experience Base
foreach memory hierarchy level M
select unmarked data structure D and loop L
s.t. D has maximum reuse, carried by L
if (level == register)
  make L innermost and unroll L
else
  permute & tile L according to reuse dimension
  generate copy variant if profitable
}
determine constraints based on D and M
[register/cache/TLB footprint analysis]
mark D

transformations
- original iteration space
  \[ s1 = \{ [k,i,j] : 1 \leq k \leq n-1 \land k+1 \leq i \leq n \land j=k+1 \} \]
  \[ s2 = \{ [k,i,j] : 1 \leq k \leq n-1 \land k+1 \leq i \leq n \land k+1 \leq j \leq n \} \]
- permute loops k and j
  \[ t1 := \{ [k,i,j] \rightarrow [0, j, 0, i, 0, k, 0] \} \]
  \[ t2 := \{ [k,i,j] \rightarrow [0, j, 0, i, 1, k, 0] \} \]
- tile loops
  \[ t1 := \{ [k,i,j] \rightarrow [0, jj, 0, kk, 0, j, 0, i, 0, k, 0] : jj=2+16 \beta \land kk = 1+128 \alpha \land i \geq 15, 2 \leq ii <= i \land kk-127, 1 <= kk <= \max(tj) = 16, tj = 128 \}
  \[ t2 := \{ [k,i,j] \rightarrow [0, jj, 0, kk, 0, i, 0, jj, 0, j, 0] : jj=2+16 \beta \land kk = 1+128 \alpha \land i \geq 15, 2 \leq ii <= i \land kk-127, 1 <= kk <= k \}

dependence analysis
reuse analysis
register model
cache model
analysis and models
Q4: Programming model?

- (Re)discover component technology and software architectures
  - Components should be “tunable”
  - Lightweight interfaces
- Unify code produced by compiler with existing libraries, new libraries, and libraries produced by domain-specific tools
  - At the component level, use the same programming model regardless of functional unit and data movement protocol
- Can “grow” a domain-specific tool (like Fortress, Telescoping Languages)
  - Learn what is effective over time
  - Evolvable and long term
Example: Optimizing Molecular Dynamics Simulation Workflows (NSF-CSR)

Collaborators: Aiichiro Nakano and Priya Vashishta (USC), Joel Saltz (OSU), Ewa Deelman, Yolanda Gil and Kristina Lerman (USC/ISI)

LACSI
Concluding Remarks

• Two core technical ideas
  – Compiler technology: Modular compilers, systematic approach to optimization, empirical search
  – Components: Tunable, XML-based interfaces, knowledge representations, more empirical search

• Focus on long-term evolutionary path

• ... And community organization
Heterogeneous systems ...

BRING 'EM ON

... But not without focus/plans/funding for software!