Programming Models

- Areas of interest to this TC
  - Code and performance portability
  - Developer productivity: tools and programming methodologies for "mere mortals"
  - Data layout and motion, multiple disjoint address spaces, SIMD length, etc.
- Relation to other TCs
  - Applications: algorithm design and selection
  - Architecture: feature roadmaps, co-design of hardware control and monitoring interfaces
  - Performance: data motion costs and analysis, system modeling

<table>
<thead>
<tr>
<th>Topic</th>
<th>Urgency</th>
<th>Duration</th>
<th>Responsiveness</th>
<th>Applicability</th>
<th>Timeline</th>
</tr>
</thead>
<tbody>
<tr>
<td>HMC Programming: Best Practices and Knowledge Transfer</td>
<td>Important</td>
<td>Medium</td>
<td>High</td>
<td>Narrow</td>
<td>Immediate</td>
</tr>
<tr>
<td>Transition Tools</td>
<td>Critical</td>
<td>Medium</td>
<td>High</td>
<td>HPC</td>
<td>Soon</td>
</tr>
<tr>
<td>Debugging and Performance Support</td>
<td>Important</td>
<td>Long</td>
<td>High</td>
<td>Broad</td>
<td>Soon</td>
</tr>
<tr>
<td>HMC and non-HMC Performance Portability</td>
<td>Important</td>
<td>Long</td>
<td>Moderate</td>
<td>Broad</td>
<td>Eventually</td>
</tr>
<tr>
<td>Expressive Programming Environments</td>
<td>Useful</td>
<td>Long</td>
<td>Moderate</td>
<td>Broad</td>
<td>Eventually</td>
</tr>
</tbody>
</table>

HMC Programming: Best Practices and Knowledge Transfer

Description

1. Quickly move knowledge from HMC research groups to code development groups (faster than developing tools)
2. Provide independent assessment of programming model technologies
3. Provide guidance on matching algorithms to hardware

Notes from Discussion
Artifacts from these activities could include:
- Reference implementations of algorithms on particular hardware configurations
- Best practices for code and algorithm design
- White papers and books
- Benchmark suites

Relations to other TCs

Applications: collaborate on design of architecture-aware algorithms
Libraries: review and preserve best practices
Architecture: co-design of algorithms, architectures, and programming models

Related Projects

CUDA Zone
Programming Motifs
MAGMA project
Machine-specific training programs

Transition Tools

Description

1. Tools to facilitate the refactoring of existing code bases to new programming paradigms
2. Tools for identifying acceleration opportunities

Notes from Discussion

- Language interoperability is a crucial component for incremental adoption of HMC features in existing codes.
- Features such as compiler or preprocessor directives are probably the quickest path to adoption

Relations to other TCs

Applications: survey of existing code bases to determine requirements and priorities for transition tools
Performance: modeling of systems to support architecture-aware code generation and rewrite capabilities

Related Projects

- Compiler directives
  - OpenMP, performance-oriented pragmas
• Language extensions and APIs
  ◦ CUDA, OpenCL, MPI
• Language translation
  ◦ C-to-CUDA, C-to-FPGA
  ◦ Note: moving from a restricted language to a general purpose language is much easier
• Performance analysis and modeling tool extensions
  ◦ ROSE, TAU

Debugging and Performance Support

Description

1. Provide capability to access debugging and performance data on HMC hardware and runtime
2. Research correlation of data from heterogeneous hardware components
3. Bridge the semantic gap between low-level data and high-level programming models

Notes from Discussion
An artifact from this activity might be a standardized interface between architectural features and higher-level tools. This would provide better tool portability across multiple architectures.

Relations to other TCs

Architecture: collaboration on two-way exchange of debugging and performance information.
Performance: determining how to strengthen the link between programming models and analysis tools

Related Projects

• Consumers of information
  ◦ NVIDIA Nexus
  ◦ vampir
  ◦ oprofile
  ◦ TAU
  ◦ TotalView
  ◦ Allinea DDT
  ◦ Charm++
• Interfaces
  ◦ PAPI

HMC and non-HMC Performance Portability

Description
1. Research how to provide performance portability across HMC and non-HMC (e.g. multicore) architectures without branching a code base.
2. How to model and develop code for explicitly-managed memory hierarchies and disjoint address spaces.

Notes from Discussion
- What are the implications of maintaining multiple code bases?
  - V&V, feature mismatches between branches, etc.
- What breadth of application space?

Relations to other TCs

Applications: what is "acceptable" performance, and when is this capability needed?
Architecture: are there opportunities for compatibility or general-purpose feature additions in HMC hardware that would ease portability

Related Projects

- Platform agnostic:
  - OpenCL
- Accelerator-to-general purpose:
  - MCUDA, CUDA-Fortran
- Autotuning

Expressive Programming Environments

Description

1. Reduce effort to utilize HMC hardware configurations
2. Capture developer's intent in a more declarative way (front-end)
3. Translate intent to HMC targets (back-end)

Notes from Discussion

Relations to other TCs

Applications: co-design of declarative programming environments

Related Projects

- Scientific computing environments
  - MATLAB, Mathematica, Maple, etc.
- FPGA Workflow
  - LabVIEW, C2H, MATLAB-to-FPGA
- High-level programming environments, languages, libraries
  - Python: Copperhead, SciPy
  - Thrust C++ CUDA library
  - UPC, Titanium
• HPCS Languages
  ◦ X10, Chapel
• Domain specific languages