Milieu Approach for Software Development for the PCA Program

Yoginder Dandass
Mississippi State University

Anthony Skjellum
Charles Summey
Hong Yuan
MPI Software Technology, Inc.

Ted Bapty
Vanderbilt University
Ben Abbott
Southwest Research Institute
Organization

• Goals
• Software Architecture
• Build Chain
• Metadata
• Dynamic Resource Management
• Software Components
• Prototype Results
• Future Work
• Conclusion
Goals of the Project

• Study and prototype a modeling language for streaming and threaded resources

• Study and prototype techniques for design space exploration in order to enable PCA scheduling

• Study and prototype techniques for system synthesis and generation

• Contribute findings to the MSI forum
Organization

- Goals
- **Software Architecture**
- Build Chain
- Metadata
- Dynamic Resource Management
- Software Components
- Prototype Results
- Future Work
- Conclusion
SAPI, SAAL, VM
(C/C++ Application)

Applications

Services (e.g., MPI, VSIPL): C/C++ API

Services: implementation

O/S: machine independent interface (API)

O/S: machine independent implementation

VM - machine independent interface (API)

VM - machine dependent implementation

Hardware interface (registers, ports, etc.)
Organization

- Goals
- Software Architecture
- Build Chain
- Metadata
- Dynamic Resource Management
- Software Components
- Prototype Results
- Future Work
- Conclusion
Current Technology Build Chain

Component Model & High-Level Metacode

bound & unbound metadata

source code

Component Metacode

Mission Compiler / Metacode Processor

Code Compiler

binary

bound metadata unbound metadata
1st Generation PCA
Build Chain

- Component Model & High-Level Metacode
- Mission Compiler / Metacode Processor
- Resource Mappings
- Code Compiler

Component Metacode

bound metadata
unbound metadata

DARPA

Mississippi State University

MPI Software Technology
Interaction of Compilers and PCA Build Chain Tools

Metacode

High-Level Scheduling, Splitting, Merging, Mapping Tools

Source code groups
Apportioned Resources

Code Translator and Optimizer
(splits and merges code for optimization, and maps code to assigned resources)

Translated Code
Metadata

Performance Metadata

API
Organization

- Goals
- Software Architecture
- Build Chain
- **Metadata**
- Dynamic Resource Management
- Software Components
- Prototype Results
- Future Work
- Conclusion
Metadata

• Describes functionality
  – Machine readable
  – Source code & language details
  – Interface and template parameters

• Known resource requirements

• Known performance capability

• Resource and Performance descriptors:
  – Static table
  – Database query
  – Parametric functions
  – Source code in high-level scripting language
Determining Metadata

- Derive requirements and capabilities from analysis of source/machine code and hardware capability (Compilers can provide this service)
- Execute code on hardware to verify execution model
- Refine model for every hardware option
  - Analytical modeling will be complex
  - Simplifying assumptions may make model inaccurate
- Construct database of known values
  - Could be large
  - Will be accurate for known hardware configurations
  - Interpolate/extrapolate
- Construct a predictive model
  - Math functions
  - Perl scripts
XML Metacode

```
<component name="FIR">
  <optimizedsection cpu="G4" dtype="double" pmode="threaded">
    <source lang="C">
      ....
    </source>
  </optimizedsection>
  <optimizedsection cpu="G4" dtype="float" pmode="streaming">
    <source lang="C">
      ....
    </source>
  </optimizedsection>
</component>

<component name="componentA">
  <source lang="C" pmode="threaded">
    -------------------
    -------------------
    -------------------
  </source>
</component>

<constantdata def type="float" label="pi">
  3.141593
</constantdata>

<variable assignment type="double" label="accum">
  <constant data ref label="pi"/>
  <call component name="FIR" dtype="double" cpu="G4"/>
  <operation optype="multiply"/>
</variable assignment>
```
Organization

- Goals
- Software Architecture
- Build Chain
- Metadata
- **Dynamic Resource Management**
- Software Components
- Prototype Results
- Future Work
- Conclusion
Dynamic Resource Management

• Needed to account for unknown events
  – Faults
  – All possible scenarios may be impossible to predict
  – May be prohibitive to generate static resource & component mappings for all scenarios

• Reduce search space by pre-selecting candidate components at build time
Application Initiated Morphs

- Application requests resource reconfiguration within allocated resources
  - May be compiler generated
  - May be explicitly coded
- Application explicitly requests new mode (components) within allocated resources
- Application explicitly requests additional resources or releases resources
- Application explicitly requests new mode (components) requiring different resources
System Initiated Morphs

• System modifies allocated resources transparently (without affecting applications’ components)
• System moves application components to execute on a different set of equivalent resources
• System modifies resources allocated to components of an application
• System modifies components and resources allocated to an application
Application/System Morph
Notification and Metadata Interface

Application

Metadata Interface

Application Morph Req.

Metadata Interface

System Morph Notification
 (callback or message-based)

Runtime System

Synchronous Morphs

Hardware

Asynchronous Morphs
Organization

- Goals
- Software Architecture
- Build Chain
- Metadata
- Dynamic Resource Management
- **Software Components**
- Prototype Results
- Future Work
- Conclusion
Component Concepts

- PCA components are hierarchical
- Design and implementation alternatives are explicitly represented in the hierarchy at any level
- A compound component may encapsulate concurrency between lower-level components
- PCA Application is a component composed from a collection of components, component clustering specifications, abstract VM specifications, and component-VM element mappings
- PCA Mission is composed of PCA Applications, VM instantiation, component to physical VM element mappings, mode-change rules, and mission constraints that govern application behaviors
Component Options

Application

X
A
B

Y
C
D
E

A_1 A_2 A_3 A_4 A_5
B_1 B_2 B_3

X_1
A_2
B_1

X_2
A_3
B_2

Y_1
C_1
D_1
E_3

Y_2
C_3
D_2
E_1

Y_3
C_1
D_2
E_4
Component Boundaries
Organization

- Goals
- Software Architecture
- Build Chain
- Metadata
- Dynamic Resource Management
- Software Components
- **Prototype Results**
- Future Work
- Conclusion
Step 1: Application Modeling

- System design using GME2000
  - Graphical modeling tool
  - Graphically design component hierarchy
- Define design alternatives
- Specify metadata for each component
- Export model using XML format
Step 2: Model Processing

- Parse the XML representation of the model
- Traverse model component hierarchy and retrieve metadata specification of each component
- Dynamically generate makefiles for intermediate code generation and compilation
Step 3: Intermediate Code

- “High-level Compiler”
- Generate the intermediate code for IRT system
- Matlab is end-user programming environment
- C is the intermediate environment
- Translate IRT source from Matlab to C
Step 4: Binaries

- “Low-level Compiler”
- Compile the intermediate code (C code)
- Generate architecture-specific executable binaries
- Metadata is expected to be interpreted and generated for resource management at this level
Scheduling Problem

- Non-preemptive schedules for parallel soft real-time applications represented as DAGs
- Metadata specifies execution time probability distributions of tasks (computation and communication)
- Metadata specifies task precedence
Schedules

\[ v_0 \leq (4, 1/3)(5, 1/3)(6, 1/3) \]
\[ v_1 \leq (7, 1/3)(8, 1/3)(9, 1/3) \]
\[ v_2 \leq (1, 1/3)(2, 1/3)(3, 1/3) \]
\[ v_3 \leq (2, 1/3)(3, 1/3)(4, 1/3) \]
\[ v_4 \leq (7, 1/3)(8, 1/3)(9, 1/3) \]

Time

\[
\begin{array}{cccccccccccccccc}
1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 & 12 & 13 & 14 & 15 & 16 \\
\hline
p_0 & v_0 & \text{(v_0, v_3)} \\
&s_0 & & \\
p_1 & v_1 & v_3 \\
&r_1 & (v_0, v_3) \\
p_2 & v_2 & \text{(v_2, v_4)} \\
&s_2 & \text{(v_2, v_4)} \\
\end{array}
\]

- \( p_n \): processor \( n \)
- \( s_n \): outgoing communication link at processor \( n \)
- \( r_n \): incoming communication link at processor \( n \)
Scheduling Options

---

**Deadline Probability vs. Jitter**

- JC = 0.00, JF = 1
- JC = 0.10, JF = 1.13564
- JC = 0.20, JF = 1.29645
- JC = 0.30, JF = 1.50579
- JC = 0.40, JF = 1.78113
- JC = 0.50, JF = 2.15417
- JC = 0.60, JF = 2.72496
- JC = 0.70, JF = 3.6519
- JC = 0.80, JF = 5.34837
- JC = 0.90, JF = 9.35711
- JC = 1.00, JF = 28.0602

---

**Compression**

- JC = 0.00, JF = 28.0602
- JC = 0.10, JF = 9.35711
- JC = 0.20, JF = 5.34837
- JC = 0.30, JF = 3.6519
- JC = 0.40, JF = 2.72496
- JC = 0.50, JF = 2.15417
- JC = 0.60, JF = 1.78113
- JC = 0.70, JF = 1.50579
- JC = 0.80, JF = 1.29645
- JC = 0.90, JF = 1.13564
- JC = 1.00, JF = 1
Organization

- Goals
- Software Architecture
- Build Chain
- Metadata
- Dynamic Resource Management
- Software Components
- Prototype Results
- Future Work
- Conclusion
Future Work - I

• Extend build chain demonstration for a parallel application using MPI on a cluster of dual Xeons with Linux
  – GME2000 metamodel
  – Multiple component implementations with parametric/predictive metadata
  – Metacode parsing and processing
  – Code generation, compilation, and linking
  – Application initiated software morphs
  – Fault handling
Future Work - II

• Continue to contribute to the MSI forum
  – Metamodelling specification
  – Metadata specification
  – APIs
    • MPI
    • VSIPL
  – Compiler and build chain tool interaction
  – VM specifications
Organization

- Goals
- Software Architecture
- Build Chain
- Metadata
- Dynamic Resource Management
- Software Components
- Prototype Results
- Future Work
- Conclusion
Conclusion

• Conceptual architecture for PCA software
• Developed prototype build chain tools
  – Matlab
  – C++, MPI, VSIPL
• Working with the MSI forum to specify
  – Software interfaces
  – Metadata elements and organization
  – Component organization
  – Tool interactions