D-TEMP: A Design Tool for Embedded Systems on Single-Chip Multiprocessors

Kenneth E. Hoganson
Kennesaw State University, 1000 Chastain Road, Kennesaw, GA, 30144, USA
khoganso@kennesaw.edu, 770-386-8414, 770-423-6731 (fax)

ABSTRACT – This paper presents the development on a system design tool (D-TEMP) intended to assist designers in constructing systems for multiprocessor platforms. The target application domain is developing embedded systems for single-chip multiprocessors. Previous work has shown that the design of some real-time constrained systems can be simplified if the target platform is a multiprocessor, exchanging hardware complexity for human design time.

The design tool utilizes a graphical drag-and-drop object-based user interface for specifying the application and platform topology and characteristics. The mapping of application to hardware platform is done through an interpreted pseudo-code language that is used to capture application-specific knowledge, and to guide the search for optimal mappings, and to allow user control of the mapping process.

Keywords: Parallel, Real-Time, Embedded, Design, Knowledge-based, Design-Tool.

1. Introduction
Embedded systems engineers will soon have single chip multiprocessor available that are inexpensive enough to serve as the platform for embedded systems [5]. Multiprocessors offer two distinct advantages to engineers and designers: 1) Speeding up processing through parallel processing. This yields a secondary benefit due to the faster processing of instructions, which can ease the design process for hard real-time systems, by allowing more functions to be completed in a given time. 2) Distributing hard real-time constrained tasks to independent processors can ease the overall design complexity by guaranteeing performance and behavior for critical functions [4].

The optimal mapping of system to multiprocessor can be a complex task. Particularly, for a new system design, where both the hardware design and software structure may not be fixed, the number of possible configurations tends toward exponentially increasing complexity. Yet, there is a need to incorporate into the design process, considerations about how the potential concurrent HW and SW structures will integrate and interact, which determine overall performance and critical timing. The integration of these considerations into the system analysis and design process has been previously presented [4].

Efficient mapping of systems onto multiprocessors has been explored, from the point-of-view of maximizing parallel speedup [1],[3], and hardware architectures that facilitate efficient mapping [2]. The problem balances two main considerations [3]: 1) Maximizing the Degree of Parallelism (DOP) of the application, which is the number of concurrently executing parallel software components. This maximizes the potential parallel speedup. 2) Minimizing the communication overhead, delays and latency between communicating parallel software components, which works to maximize the realized parallel speedup. Increasing the DOP tends to increase the frequency and size of data that must be exchanged between parallel components of an algorithm. The time required for this transfer (whether through message passing

This work is being supported by Georgia’s Yamacraw Initiative. www.yamacraw.org
or shared-memory) is non-computational overhead that detracts from the realized performance gain.

Considerations that determine the efficient mapping of software to hardware should be included in the design process at an early level. But because of the complexity of these issues (both in terms of the state-space of possible HW and SW architectures and their interactions), and the complexity of analyzing and comparing alternatives, there is a need for a design tool to support engineers in this task. The tool under development is called D-TEMP, for Design Tool for Embedded MultiProcessors.

This paper presents the current design tool interface elements, the hardware and software data acquisition and analysis strategies, the mapping algorithms, and performance evaluation, metrics.

2. Performance Measurement
A metric for measuring a reduction in system development has been presented [4] called Temporal-Constraint. This metric quantifies the enhanced ease in meeting hard real-time constraints resulting from distributing time-constrained tasks to individual processors. It is a ratio of the worst case/critical path time values. The numerator is the worst-case/critical path time before distributing across multiple processors, and the denominator is the worst case/critical path after distributed hard real-time tasks to multiple processors (Equation 1).

\[
\text{Temporal - Constraint} = \frac{\text{Worst} - \text{Case Time Required on Uniprocessor}}{\text{Max}[\text{Worst} - \text{Case Time Required on Multiprocessor}]}
\]  

(1)

The second metric to be used in evaluating alternative designs is the well-known parallel speedup ratio of serial time over parallel time (Equation 2).

\[
\text{Speedup} = \frac{\text{Serial - Time}}{\text{Parallel - Time}}
\]  

(2)

3. D-TEMP Design and Interface
D-TEMP allows the user to enter a Platform Characteristics and Communication Topology (PCCT) using pull-down menus and drag-and-drop objects. The user/designer also enters the Application Communication Topology (ACT) in a similar manner. The mapping analysis can then be initiated, and a high-level simulation can be executed of bottleneck analysis. The designer may modify either the ACT or PCCT and rerun the analysis or simulation, and has the ability to customize both the mapping rules, and mapping evaluation or ranking rules. Diagram 1 shows the pull-down menus of the user interface.

A wide range of platform architectures can be modeled including: two-processors with shared memory, single clustered machines, two levels of interconnect and clustering, multi-bus multiprocessors, and distributed architectures (over a network or the Internet). For each interconnect layer, the user must specify latency, bandwidth, and contention mechanism (if present) for bottleneck/capacity simulation analysis.

The tool software design decomposes into a natural set:

- A set of objects and data-structures to hold the ACT and PCCT.
- A user interface, with drag-and-drop.
- Mapping components, list traversals, mapping strategies
- Pseudo-language/preprocessor interpreter to allow design engineers to manipulate the mapping rules and strategies.
- Mapping evaluation algorithms
- High-level simulation component

3.1 Drag-and-Drop Objects
Diagram 2 shows a subset of the objects available. Each object has a properties box that appears when the user right-clicks on the object. This data is required for the mapping analysis. The graphical system captures:

- The type of module involved: i.e. interrupt service routine, Real-Time Operating System (RTOS), thread, and other modules indicated with different shaped objects.
- The communication topology.
- Parameters exchanged and communication mechanism.
• Flow-of-control information about module communication, used to simulate the system for bottleneck and utilization analysis.

3.2 Steps to Use the tool
The following steps describe a typical tool use scenario:

1. User enters Platform Characteristics and Communication Topology (PCCT), using GUI interface and pull-down menus with drag-and-drop objects. Object characteristics are edited in an annotation pop-up properties box associated with each object. The PCCT is stored as a notated graph.

2. User enters Application Communication Topology (ACT), using GUI interface and pull-down menus with drag-and-drop objects. Object characteristics are edited in an annotation pop-up associated with each object. The ACT is stored as a notated graph.

3. The mapping process traverses the ACT to discover mapping constraining data: i.e.

   “main” module(s) and hard real-time constrained tasks or threads, user-specified mappings of specific components to HW.

4. The application uses rule-based knowledge to explore the state-space of possible mappings of the constraining components. The knowledge encoded as rules makes the state-space exploration more efficient both in terms of time to process, and generating favorable mapping.

5. The application evaluates each mapping of the constraining components by calculating the temporal-constraint and parallel speedup metrics, and ranks the mappings according to rule-encoded knowledge (i.e. a weighted average of the two metrics).
6. The tool then completes the mapping of the entire ACT for the top five mappings.
7. The application then evaluates each complete ACT mapping to PCCT by re-computing the metrics, and re-ranks the mappings.
8. The “best” mapping is displayed graphically on the user’s monitor, along with the calculated metrics, and a bottleneck analysis.
9. The designer may elect to run a high-level simulation of the mapping, to search for bottlenecks, and utilization of processors and communication links. The results are presented visually, and the user may query each object in the PCCT (click on it) to see data from the high-level simulation of component utilization and possible bottlenecks.
10. The designer may elect to have the tool make suggestions for modifying the ACT, or reconfigure the PCCT or ACT to improve performance.
11. The designer may make modification to the ACT or PCCT and re-run the analysis or simulation.
12. Also, the designer has may enter additional rules, or modify existing rules.

4.0 Mapping Knowledge Structure
The core of this application is its mapping analysis ability or knowledge. Two general approaches to incorporating this “intelligence” were considered.

Algorithmic and brute-force searching methods of mapping, that explore and evaluate
every possible mapping, and select the highest ranked mapping. Real-world applications and problems may have additional constraints that are unique to a specific application, which may be difficult to insert in an algorithmic mapping. In the parallel-embedded system domain, one can imagine that the RTOS might be constrained to run on a specific processor, perhaps based on location, memory constraints, human access to the machine, etc. Or a device that generates interrupts to a local processor, which is part of a distributed (over a LAN) system, requires that the task that evaluates and controls that device be physically local to that device. In either case, it may be difficult to manipulate the data-set that the algorithm will work on to force those specific mappings, and it may be unrealistic to require engineers to spend the time to understand the underlying algorithms sufficiently to apply manipulated or weighted data appropriately. Likewise, it is not reasonable to require the user to search through a set of possible solutions looking for the best one that also satisfies external constraints.

The alternative adopted has a reasonably convenient and clear method for engineers to enter application specific constraints. After considering both forward and backward chaining rule-based systems, a pseudo-code interpreter for storing mapping meta-knowledge was adopted. Embedded systems engineers are comfortable with programming, and can take advantage of the ability to customize the mapping algorithms, and develop new rules for different problem domains. The interpreted pseudo-code being implemented is a verbose English-like structure, similar to many rule-based systems, with some similarities to BASIC. It can be translated directly to methods defined for and manipulating underlying objects. The translation process then is similar to a preprocessor. In this way, the details of the underlying objects and structures are hidden from the user, both to simplify the interface, and for good software design supporting the principle of information hiding.

The mapping algorithms written in this pseudo-code like language translate to a set of methods that can perform operations on the graphs/data-structures that hold the ACT and PCCT. These methods can find specific objects, specific types of objects, or objects with specific properties by traversing lists of objects. Iterations through the graph can be performed, searching, selecting and mapping SW objects to HW. Special map ranking rules using these “public” methods and control structures can be created and customized and controlled by the engineer.

Consider Rule 2 of Table 2. It encompasses multiple operations on the data structures:
1. A search for a Real-Time Operating System (RTOS) component in the ACT.
2. An iteration over the possible mappings of RTOS to processors (PCCT), executing rules 2.1 and 2.2, for each possible mapping, each of which contain iterations through the data structures themselves. Fan-in and Fan-out refer to the number of other SW components that call, and are called by this object.

In this way, a great deal of processing complexity on the ACT and PCCT graphs can be controlled and specified without requiring the engineer to understand the details of the implementations of the data structures or the mechanics of operations on the data structures.

Four categories of meta-rules/algorithms are being developed:
1. Algorithms to explore the PCCT and obtain identify specific features of the PCCT that will affect the mapping. This processing will occur prior to the actual mapping.
2. Algorithms to map the ACT to PCCT, i.e. Rule 2 through 2.2. in Table 2.
3. Mapping ranking algorithms or rules, i.e. Rule 1. in Table 1.
4. ACT suggestion analysis, to offer possible modifications to the ACT, based on opportunities to reconfigure the SW design to improve the mapping without compromising the SW architecture.

<table>
<thead>
<tr>
<th>Table 1. Mapping Algorithms (pre-defined)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Example Mapping Ranking Rule</td>
</tr>
<tr>
<td>1. MapScore = Speedup + TimeDilation</td>
</tr>
<tr>
<td>Example Mapping Algorithms</td>
</tr>
<tr>
<td>3.1 Map to HWOBJ in ascending order of sum(fan-in + fan-out)</td>
</tr>
<tr>
<td>3.2 For All unmapped map-constraining objects, in ascending order of sum(fan-in + fan-out) map to processor with the min(sum(Fan-in + Fan-Out))</td>
</tr>
</tbody>
</table>
performance, real-time constraints, etc. These rules will access a fourth level or programming, the methods that act upon the ACT and PCCT graphs structures directly.

The mapping algorithms focus on creating large time margins for guaranteeing hard real-time constraints, minimizing communication frequency, latency, and overhead, maximizing the improvement in the Temporal-Constraint metric and maximizing parallel speedup.

4.1 Mapping Language Constructs
The following list of user-accessible commands can be used to generate and edit mapping-knowledge rules:

- TRAVERSE (list)
  - Statements
  - ENDTRAVERSE
- IF (object-in-list) property (=,<,>) value THEN
  - Statements
  - ENDIF
- IF (object-in-list) property (NOT) MAPPED
  - Statements
  - ENDIF
- ADD (object-from-list) TO list;
- MAP (object-from-list) USING mapstrategy;

Mapping options:
- USER-SPECIFIED
- FIRST-FIT
- BEST-FIT WEIGHTED (statement that specifies a weighted-average of property values.

4.2 Data Capturing Objects
The following objects are predefined for use in capturing information from the user about the application and target platform.

- SWOBJ
  - ID-Number?
  - Name/description
  - Type
  - Mapped-to
  - Calls/sends-to list
  - Called/receive from list

Table 2. Mapping “Rules”

<table>
<thead>
<tr>
<th>Logical Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Search for critical objects, (a simple linear examination of SW objects in the ACT: RTOS (often specified on a processor), or OS REAL-TIME constrained entities Hardware-coupled objects (often specified on a processor) User-specified critical types And map those objects as specified by the user, or finding a best location based on a weighting algorithm.</td>
</tr>
<tr>
<td>Then map the non-critical SW objects.</td>
</tr>
<tr>
<td>Control structure is a loop through the objects If SWObject.type = critical-type then Add to critical list; If not-user-mapped then Find the best HW object (Linear Search through HW-objects) where SWObject.mappable=HWObject.type Use rule knowledge to weight “best” Map SW-object to best HW-object Loop through the SW-Objects If not already mapped, map the object using a first-fit strategy.</td>
</tr>
</tbody>
</table>

User-Edit-able Preprocessed Language

1. TRAVERSE SWOBJ /*implies SWOBJ for next statements*/
   - IF TYPE = RTOS THEN
     - ADD TO CRITICAL-LIST;
     - IF USER-SPECIFIED MAPPING THEN
       - CREATE MAPPING;
       - MAP USING USER-SPECIFIED;
     - ELSE
       - TRAVERSE HWOBJ;
     - IF TYPE = PROCESSOR
       - CREATE MAPPING;
       - MAP SWOBJ TO HWOBJ;
     - ENDIF
   - ENDIF
   - ENDIF
   - ENDTRAVERSE

2. TRAVERSE MAPPING-LIST

2.1 TRAVERSE SWOBJ
   - IF TYPE = CRITICAL AND NOT MAPPED THEN
     - IF USER-SPECIFIED MAPPING THEN
       - ADD TO CRITICAL-LIST;
       - MAP USING USER-SPECIFIED;
     - ENDIF
   - ENDIF
   - ENDTRAVERSE

2.2 TRAVERSE CRITICAL-LIST
   - IF NOT MAPPED THEN
     - MAP USING BEST-FIT;
   - ENDIF
   - ENDTRAVERSE

2.3 TRAVERSE SWOBJECTS
   - IF NOT MAPPED THEN
     - MAP USING FIRST-FIT;
   - ENDIF
   - ENDTRAVERSE
Lists of each type SWOBJ (Software object, part of the ACT), HWOBJ (hardware object, part of the PCCT), COMOBJ (communications object), are predefined for use in data acquisition and mapping.

5. A Brief Example
Diagram 2 illustrates example objects, but can also be used as an example ACT to be mapped to PCCT. The following steps illustrate the tool’s mapping strategy from the user-entered data of Diagram 2, generating the mapping of Diagram 3 in the process:

1. Locate a RTOS in the ACT (SWOBJ), and map it.
2. Locate a “main” or parent module (if present) and map it. To minimize communication, the default mapping is to the processor running the RTOS (though the user could specify a different location when entering the ACT).
3. Locate hard real-time constrained tasks, and map them. They are both mapped to the same processor, to balance the load equitably with the processor running the RTOS and the “main” or parent module.
4. Map other components, in this case, the interrupt interface routine for a HW device, while balancing the load on processors and fulfilling other constraints. For this example, the second processor greatly eases the difficulty in implementing code for Thread1 and Thread2 that execute in the worst-case within the time constraints, thus easing the system design process, saving design time and effort.

6. Conclusion
A tool for mapping applications to multiprocessors is being developed for general use, with specific application to embedded systems on single-chip multiprocessor platforms. The underlying premis is that utilizing multiprocessors for hard-real-time constrained systems offers advantages that speed up and simplify the design process. Evaluation metrics and incorporation of the tool into the design process were previously explored. Presented here are details on the user interface, data acquisition, and details on the strategy and mechanics of “intelligence” that governs the mapping ACT to PCCT.

A secondary use of the tool is as a teaching tool for embedded systems, parallel architectures and algorithms, and distributed computing.

REFERENCES