Multiprocessor Allocation and Scheduling using Advanced Optimization Technology

L. Benini, M. Milano, D. Bertozzi*  
M. Lombardi, A. Guerri, M. Ruggiero  
Università di Bologna,  
*Università di Ferrara

Embedded Application pull

- 1TOPS/W
- 100GOPS/W
- 5 GOPS/W

[IMEC]


Year of Introduction
MPSoC – 2005 ITRS roadmap

[Martin06]

MPSoC Platform Evolution

Applications ➔ Software opt. ➔ Middleware, RTOS, API, Run-Time Controller

Mapping & scheduling V, Vt, Fclk, Il

Bus based Multi Proc

Local Memory hierarchy

Power Test Mgmt

Net Int
Design as optimization

- Design space
  The set of “all” possible design choices
- Constraints
  Solutions that we are not willing to accept
- Cost function
  A property we are interested in (execution time, power, reliability…)

When & Why Offline Optimization?

- Plenty of design-time knowledge
  - Applications pre-characterized at design time
  - Dynamic transitions between different pre-characterized scenarios
- Aggressive exploitation of system resources
  - Reduces overdesign (lowers cost)
  - Strong performance guarantees

Applicable for many embedded applications
The problem of allocating, scheduling for task graphs on multi-processors in a distributed real-time system is **NP-hard**.

New tool flows for **efficient** mapping of multi-task applications onto hardware platforms.
**Target architecture**

- **Homogeneous computation tiles:**
  - ARM cores (including instruction and data caches);
  - Tightly coupled software-controlled scratch-pad memories (SPM);
  - AMBA AHB;
  - DMA engine;
  - RTEMS OS;
  - Power models for 0.13µm power models (STM)
  - Variable Voltage/Frequency cores with discrete (Vdd,f) pairs
  - Frequency dividers scale down the baseline 200 MHz system clock
  - Cores use non-cacheable shared memory to communicate
  - Semaphore and interrupt facilities are used for synchronization

**Application model**

- **Task graph**
  - A group of tasks T
  - Task dependencies
  - Execution times express in clock cycles: $WCN(T_i)$
  - Communication time (writes & reads) expressed as: $WCN(W_{T_iT_j})$ and $WCN(R_{T_iT_j})$
  - These values can be back-annotated from functional simulation or computed using WCET analysis tools (e.g. AbsINT)
  - Node type
    - Normal; Fork, And; Branch, Or

![Task graph diagram]

![Application model diagram]
Each task has three kinds of memory requirements:
- Program Data
- Internal State
- Communication queues

Task storage can be allocated by Optimizer:
- On the local SPM
- On the remote Private Memory

Communicating tasks might run:
- On the same processor → negligible communication cost
- On different processors → costly message exchange procedure
**Application Development Flow**

CTG → Characterization Phase → Application Profiles → Optimization Phase → Optimal SW Application Implementation → Platform Execution

**Optimization framework**

- Deterministic & stochastic task graphs
- Constraints
  - Resources: computation, communication, storage
  - Timing: task deadlines, makespan
- Objective functions
  - Performance (e.g. Makespan)
  - Power (energy)
  - Bus utilization
- General modeling framework → highly unstructured optimization problems
  - No black-box/generic optimizer can solve them efficiently
- **We developed a flexible algorithmic framework which is tuned on specific problems**
Logic Based Benders Decomposition

Decomposes the problem into 2 sub-problems:
- Allocation & Assignment (& freq. setting) → IP
  - Objective Function: E.g.: minimizing energy consumption during execution and communication of tasks
- Scheduling → CP
  - Objective Function: E.g.: minimizing energy consumption during frequency switching

Computational scalability

- Simplified CP and IP formulations
- Hybrid approach clearly outperforms pure CP and IP techniques
- Search time bounded to 1000 sec.
  - CP and IP can found a solution only in 50%- of the instances
  - Hybrid approach always found a solution
Computational Scalability

Stochastic task graphs, mapping & scheduling & min bus usage

Solution time for slightly structured instances

- Hundreds of decision variables
- Much beyond ILP solver or CP solver capability

Optimality gap

Comparison with heuristic 2-phase solution (GA)

“timing barrier”

gap significant when constraints are tight
The abstraction gap between high level optimization tools and standard application programming models can introduce unpredictable and undesired behaviours.

Programmers must be conscious about simplified assumptions taken into account in optimization tools.

**Validation of optimizer solutions**

- MAX error lower than 10%
- AVG error equal to 4.51%, with standard deviation of 1.94
- All deadlines are met
MAX error lower than 10%;
AVG error equal to 4.80%, with standard deviation of 1.71;

---

**GSM Encoder**

**Task Graph:**
- 10 computational tasks;
- 15 communication tasks.

Throughput required: 1 frame/10ms.

With 2 processors and 4 possible frequency & voltage settings:

<table>
<thead>
<tr>
<th>Without optimizations:</th>
<th>With optimizations:</th>
</tr>
</thead>
<tbody>
<tr>
<td>50.9µJ</td>
<td>17.1 µJ</td>
</tr>
</tbody>
</table>

- 66.4%
Challenge: programming environment

- A software development toolkit to help programmers in software implementation:
  - a generic customizable application template \(\rightarrow\) OFFLINE SUPPORT;
  - a set of high-level APIs \(\rightarrow\) ONLINE SUPPORT in RT-OS (RTEMS)
- The main goals are:
  - predictable application execution after the optimization step;
  - guarantees on high performance and constraint satisfaction.
- Starting from a high level task and data flow graph, software developers can easily and quickly build their application infrastructure.
- Programmers can intuitively translate high level representation into C-code using our facilities and library

Example

- Number of nodes (e.g 12)
- Graph of activities
- Node type
  - Normal, Branch, Conditional, Terminator
- Node behaviour
  - Or, And, Fork, Branch
- Number of CPUs: 2
- Task Allocation
- Task Scheduling
- Arc priorities

//Node Type: 0 NORMAL; 1 BRANCH; 2 STOCHASTIC
//Define N_CPU:
//Define Node Behaviour: 0 AND; 1 OR; 2 FORK; 3 BRANCH
//Define Task on core TASK_NUMBER = {1,2,..12}:
//Define Schedule on core N_CPU = {{1,2,..12}):
//Define Node Behaviour TASK_NUMBER = {0,1,2,3}:
//Define (0,0,0..,)

<table>
<thead>
<tr>
<th>Time</th>
<th>Resources</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>B2</td>
</tr>
<tr>
<td></td>
<td>N1</td>
</tr>
</tbody>
</table>

Deadline
Relationship with RT techniques

- We can handle periodic task graphs
  - Multiple rates can be analyzed by unrolling and periodic extension
- Cannot deal with aperiodic/sporadic tasks unknown at design time
  - They would require unbounded unrolling
- Currently assumes non-preemptive scheduling

Summary & future work

- Toward a mature SDK
  - Mature programmer support (Eclipse toolkit, OpenMAX support)
  - Extend semantics (multi-rate SDF)
  - Ports on real platforms (Cell BC underway, Nomadik is under discussion)
- Optimization engine enhancements
  - Dealing with multiple use cases
  - Variable execution times
  - Aggressive communication scheduling on NoCs
  - Address preemption and sporadic tasks
Backup Slides

Allocation problem model

The objective function: minimize energy consumption associated with task execution and communication

Each task can execute only on one processor at one freq.

Communication between tasks can execute only once for execution and one write corresponds to one read

\[
\sum_{p=1}^{P} \sum_{f=1}^{M} X_{tp} = 1 \forall t \\
\sum_{p=1}^{P} \sum_{f=1}^{M} W_{ijfp} \leq 1 \forall i, j \in T \\
\sum_{p=1}^{P} \sum_{f=1}^{M} R_{ijfp} \leq 1 \forall i, j \in T \\
\sum_{p=1}^{P} \sum_{f=1}^{M} (W_{ijfp} - R_{ijfp}) = 0 \forall i, j \in T
\]

\[OF = En_{\text{Comp}} + En_{\text{Read}} + En_{\text{Write}}\]
Allocation problem model

Communication energy for Writes to shared memory. Writes carried out at the same frequency of the task:

\[ E_{\text{Write}} = \sum_{p=1}^{P} \sum_{f=1}^{T} \sum_{t=1}^{T} (X_{i,p,f} WCN_{\text{LocRij}} + W_{i,p,f} (WCN_{\text{RemRij}} - WCN_{\text{LocRij}})) E_f \]

Communication energy for Reads from shared memory. Reads carried out at the same frequency of the task:

\[ E_{\text{Read}} = \sum_{p=1}^{P} \sum_{f=1}^{T} \sum_{t=1}^{T} (X_{i,p,f} WCN_{\text{LocRij}} + R_{i,p,f} (WCN_{\text{RemRij}} - WCN_{\text{LocRij}})) E_f \]

\[ OF = E_{\text{Comp}} = E_{\text{Write}} + E_{\text{Read}} \]

Computation energy for all tasks in the system:

Scheduling problem model

Duration of task i is now fixed since mode is fixed:

\[ Dur_w = \frac{WCN_{\text{max}}}{freq_w} \]

- Five phases behaviour:
  - INPUT=input data reading;
  - EXEC=compuation activity;
  - OUTPUT=output data writing.

- Atomic activities

**The objective function:** minimize energy consumption associated with frequency switching.

- Processors are modelled as unary resource
- Bus is modelled as additive resource

\[ \text{Tasks running on the same processor at the same frequency} \]
\[ \text{Tasks running on the same processor at different frequencies} \]
\[ \text{Tasks running on different processors} \]
Queue ordering optimization

- Communication ordering affects system performances

Queue ordering optimization

- Communication ordering affects system performances
Synchronization among tasks

Non blocked semaphores