Combinatorial Optimization for Embedded System Design
Luca Benini

Work in cooperation with Michela Milano’s group

Embedded Systems

A rough definition
“Any computing system which is not a computer”
- Large variety of devices
- High performance (as they often real time applications)
- High energy efficiency (e.g. in case of battery supplied power)

Classical design approaches
- dedicated G.P. systems
- dedicated hardware

Issue: efficiency
Issue: high design cost + poor flexibility
MultiProcessor Systems on Chip

**MPSoCs** address all such problems:
- Flexibility through software (or mixed HW/SW) applications
- Performance through parallelism
- Low power consumption by using low(er) frequency cores and power saving techniques

**HOWEVER:**
- Requires proper use of the exposed resources
- Squeezing out the full power of a modern MPSoC can be definitely HARD

---

**Design productivity gap**

<table>
<thead>
<tr>
<th>Year</th>
<th>Technology</th>
<th>Chip Complexity</th>
<th>Frequency</th>
<th>Staff</th>
<th>Staff Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>1997</td>
<td>250 nm</td>
<td>43 M Tr</td>
<td>400</td>
<td>210</td>
<td>90 M</td>
</tr>
<tr>
<td>1998</td>
<td>250 nm</td>
<td>20 M Tr</td>
<td>500</td>
<td>270</td>
<td>120 M</td>
</tr>
<tr>
<td>1999</td>
<td>180 nm</td>
<td>32 M Tr</td>
<td>600</td>
<td>360</td>
<td>160 M</td>
</tr>
<tr>
<td>2002</td>
<td>130 nm</td>
<td>133 M Tr</td>
<td>800</td>
<td>800</td>
<td>360 M</td>
</tr>
</tbody>
</table>

* @ $150K / Staff Yr. (In 1997 Dollars)
Design productivity gap

1. Mixed hardware/software design deals with the issue...
   ...Provided you can design efficient software
2. Optimizing a simple application for a 6 core platform can take months...
   ...And 10s to 100s core systems are planned for the next future

Ideally

Given:  
- Input application (code)  
- Target Platform description  
Yield: optimized application

What’s the black box?
- A compiler?  
- A CAD tool (less blackish)?  
- A run time support?

OPT. APP.

FOR SURE:
This thing has to perform some automatic optimization
On-line vs off-line approaches

How should the black-box look like?
→ most likely: two distinct components

ON-LINE
Some examples you may know:
= OS level scheduler
= On-line application-to-core dispatcher (Current multi-core CPUs...)
= Out of order execution
= ...

OFF-LINE
Some examples you may know:
= Off-line code optimization (e.g. VLIW compilers)
= Memory allocation (even hand made)
= Off-line resource allocation (e.g. mixed HW/SW design)

↓ A lot of information must be a priori known
↑ Easier to provide guarantees
↑ Simple run-time support
↑ Simple hardware

↑ Adapts to dynamic workload changes
↓ Hard to provide guarantees
↓ Requires more complex run-time support
↓ May require more complex hardware
Requirements for automatic optimization

1. A formal description of the application must be available
   - Formal =
     1. can be understood by a computer
     2. can be manipulated by mathematics
   - Usually: task based models
     - Task = atomic computation unit
       (e.g. an instruction, a process, a code block...)
     - Tasks may have dependencies
     - Task have measurable “features”
       (e.g. execution time) which must be computed in some way
     - Tasks use hardware resources

2. A formal description of the platform must be available
   - Usually: resource based models
     - Resource = an “energy” provider over time
     - Each resource has a finite capacity
     - Platform = collection of resources
   Additionally, for off-line approaches:
3. A formal description of the performance metrics must be available
   - completion time (makespan)
   - throughput
   - energy consumption
   - number of bus transactions
   - ...

[Diagram with tasks and dependencies]
An example on MPSoCs

Mapping & Scheduling Problem:
- Application description
- Platform description

Through:
- Off-line optimization algorithm

Provide directives for the run-time support:
- Resource-to-task allocation
- Task scheduling decisions

Compiled code + directives + run time support = OPTIMIZED APPLICATION

An example: mapping

Mapping = allocate platform resource to tasks
E.g. Each task requires a processor for its execution and a empty device to store computation data.
An example: scheduling

Scheduling = decide when tasks must start

mapping & scheduling: solution

Once the problem is formalized, it can be solved via MANY tools

- There’s a rich literature about this
- More/less quality effective approaches
- More/less fast approaches
In this lecture

We will focus on Mapping & Scheduling for MPSoCs platforms and:

1. Show how the input for the process can be described
   - Models of Computation
   - Resource based platform model

2. Show some basic techniques for the run-time support:
   1. Scheduling techniques
   2. Mapping techniques

3. Once we know something about 1 and 2, we can tackle off-line optimization:
   - Derive a formal problem definition
   - Present solution techniques

4. Some (important) final considerations

In this lecture

This outlines a design flow:

- Define Model
- Define Run-Time Support
- Define Off-line Problem
- Model actual input
- Solve Off-line Problem
- Execute on the Run-Time support
Step 1: Application and Platform Model

Resource Based Platform Model

Although usually not specified, most mapping & scheduling approaches assume a resource based platform description model.

Basic building block: resource, i.e.
- An energy provider over time
- Has finite capacity (let this be “Cap”)
- Tasks require some amount of energy from the resource (let this be “Crj” for resource “r” and task “j”):
  - Since they start, until they end
  - Since they start
  - Until they end
  - Always
  - ...


**Resource Based Platform Model**

On a Gantt chart:

- Resource required *from start to end*

- Resource required *always*

**How to...**

**Model a processor/core?**
E.g. Single thread CPU

- \( \text{Cap} = 1 \)
- \( C_{rj} = 1 \) for each task using the processor
- Each tasks uses the resource from start to end

E.g. Dual thread CPU

- \( \text{Cap} = 2 \)
- \( C_{rj} = 1 \) for each task using the processor
- Each tasks uses the resource from start to end
How to...  

Model a storage device?  
E.g. dynamically managed storage  
- \(\text{Cap} = \text{size of the device}\)  
- \(C_{ri} = \text{amount of memory required}\)  
- Each task uses the resource from start to end  

E.g. statically managed storage  
- \(\text{Cap} = \text{size of the device}\)  
- \(C_{ri} = \text{amount of memory required}\)  
- Each task uses the resource \text{always}\)  

Storage is partitioned once for all  

How to...  

Model a BUS?  
E.g. single bus  
- \(\text{Cap} = \text{bandwidth}\)  
- \(C_{ri} = \text{required bandwidth}\)  
- Each task uses the resource from start to end  

E.g. crossbar (N cores)  
- \(N \times N\) resources  
- \(\text{Cap} = 1\)  
- \(C_{ri} = 1\)  
- Each task uses the resource from start to end
Alternative modeling styles

Example: Intel SCC (Single chip Cloud Computer)
High performance multicore system

Modeling Style A:
- resource = core
- 48 resources
- Cap = 1

Modeling Style B:
- resource = tile
- 24 resources
- Cap = 2

What to model?

Basic principles:
- A model is an abstract, simplified representing the relevant elements of a given reality
- Hence, everything that matters should be modeled

- matters = sets actual constraints on the execution
- Use of profiling or preliminary analysis for the identification
### A Practical case: CELL BE

**CELL BE** (by IBM, Sony and Toshiba)
- **Power Processing Element (PPE)**
- **Large off Chip memory (DRAM) controller**
- **8 Synergistic Processing Elements (SPE)**
  - 1 single thread proc.
  - 1 small local mem
  - high bandwidth BUS
  - DMA

- 8 unary resources
- 1 very large capacity memory
- 8 finite cap. Memories, usage start to end
- (relatively) small bandwidth usage

### A Practical case: Multiprocessor ARM

- **Single thread**
- **Small size scratchpad**
- **Large size**
- **No design choice on this**
- **A shared storage area**

- **N unary resources, usage from start to end (cores)**
- **N limited capacity resources, always used (statically allocated scratchpad)**
Applications: Model of Computation

A paradigm or model of computation (MOC) is a meta-model or "ontology" which specifies what kinds of entities, relationships, and events there are in a particular computational system.

MOC are abstract models.

MoC: Real-Time applications

We define an application in terms of concurrent and independent processes with prefixed temporal specifications.

For each Process $i$:

- Execution Time $Ex_i$
- Execution Frequency (period) $T_i$
- Deadline Execution Time $D_i$

Clearly the $k$-th execution of actor $i$:

- Starts at $k*T_i$
- MUST End by $k*T_i + D_i$

![Diagram showing real-time application model with process execution details and deadlines.]}
MoC: Task Graph

Task Graph: Each Node correspond to a task. Tasks have precedence Relations.

For each Task $j$ of the process $i$:

- **Execution Time** $E_{xj}$
- Foreach resource $r$ a requirement $C_{ij}$

Task $T_4$ MUST execute after $T_1$ and $T_2$ have finished.

Precedences = Data Dependences

Concurrent Execution

---

MoC: Synchronous Data-Flow Graph

Node (Actors) – Task
Edge - Communication Channel (FIFO)
Token – Data

Concurrency
Execution Constraint (rate)
Periodic Behavior (Repetition Vector)

Repetition Vector: [3,2,2,1]

cite
Scheduling & Mapping Techniques in the run-time support

Basic Mapping and Scheduling Problem

Application Model → Architecture Model → Platform

MAPPING

Application Platform
Basic mapping & scheduling problem

MAPPING
Resource requirements may be computed off-line by simulation

Application

Architecture

Resource conflict

Partial Order

Basic mapping & scheduling problem

SCHEDULING
The scheduling problem is in general an NP-complete problem, and algorithms for optimal scheduling in polynomial-time are known only for simple cases

The makespan is the total execution time of the application; with periodic applications the makespan is the average iteration period.

The schedules are schematically represented as a Gantt chart, which indicates the resources along the vertical axes, and time along the horizontal axis. The actors are represented as rectangles with horizontal length equal to the execution time of the actor. The left edge of each actor in the Gantt chart corresponds to the starting time of the corresponding actor.
Scheduling Strategies

Trade-off of generality against run-time overhead and implementation complexity

- Fully Dynamic
- On-line Schedule
- Fully Static
- Off-line Schedule
- Static assignment
- Quasi-static
- Self-timed
- Ordered-transaction
- Run-time overhead, implementation complexity

Scheduling Strategy: Fully Static

The exact firing time of each actor is assumed to be known at compile time.

Complete and precise knowledge of the application (tasks, precedences), of the architecture (structure, resource capacities) and of their relation (execution times, capacity requests,...).

\[ \text{Makespan} = 61\text{ns} \]

2 SingleThread Processors
2 RAM 512KB capacity
Scheduling Strategy: Fully Dynamic

All scheduling decisions are determined at run-time.

**Most general** in terms of applicability

Handle **highly dynamic program behaviour** by changing the order in which tasks are executed, and/or adjusting resources loads during run-time.

However the **cost** of such run-time scheduling decisions is very high.

Since scheduling decision are made during run-time, it is not practical to make globally optimal scheduling decisions.

**Dynamic Scheduling: incomplete approach**

Optimal multi-resources scheduling for task graph application is known to be NP-Hard problem, nad a number of heuristics have been proposed for this problem.

One of the earliest, and still popular, solutions to this problem is **list-scheduling** [Hu61].

List-scheduling is a **greedy** approach: whenever a task is ready to run, it is scheduled as soon as a processor is available to run it.

Tasks are assigned priorities, and among the tasks that are ready to run at any instant, the task with the highest priority execute first. The Priority List is computed/decided Off-Line!

Application: 9 independent processes
Architecture: 4 unary resources

<table>
<thead>
<tr>
<th>Application</th>
<th>Processing Times: [7 7 6 6 5 4 4 4]</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Processing Times: [7 7 6 6 5 4 4 4]</th>
<th>LIST SCHEDULING</th>
<th>Optimal Solution</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 2 3 4 5 6 7 8 9</td>
<td>1 2 3 4 5</td>
<td>1 2 3 4</td>
</tr>
<tr>
<td></td>
<td>7 8 9</td>
<td>7 8 9</td>
</tr>
</tbody>
</table>

**LIST SCHEDULING Optimal Solution**
Scheduling Strategy: Fully Static

The exact start time of each task is specified off-line

- No on-line resource arbitration (all possible resource conflicts are resolved off-line)
- Need for a global timing reference
- Best performance in case of fixed durations
- Strong inefficiency in case of variable task durations

Static versus Dynamic scheduling strategies

<table>
<thead>
<tr>
<th>Static</th>
<th>Dynamic</th>
</tr>
</thead>
<tbody>
<tr>
<td>Complete knowledge of the application/architecture and their relations (execution times, resource requests, ...)</td>
<td>Basic knowledge of the application (actors and precedences) and the architecture (structure and capacities)</td>
</tr>
<tr>
<td>Same Computation costs, but More time to spend in search</td>
<td>Same Computation costs, but very Limited time to spend in search</td>
</tr>
<tr>
<td>Problems with dynamic program behaviour</td>
<td>Handle highly dynamic program behaviour</td>
</tr>
<tr>
<td>Low implementation costs</td>
<td>High implementation costs</td>
</tr>
</tbody>
</table>

In the presence of compile-time accurate information about tasks, resource requests, and precedence constraints, a static scheduling approach may lead to better performance [SL93]
Scheduling Strategy: Self-timed

It works well with streaming/periodic applications.

The resource sending data writes data into a FIFO buffer, and blocks when that buffer is full; the receiver blocks when the buffer it reads from is empty.

Thus Flow control is performed at RUN-TIME!

Synchronous Data-Flow Model of Computation
Mapping Techniques

Some underlying ideas:
- Resources can be divided into classes
- Each class identifies a set of resources having the same function
- Each tasks may use one or more classes

Then mapping consists into assigning to task a resource for each class

We can distinguish three cases:
- Pure on-line mapping
- Pure off-line mapping
- Mixed on-line/off-line mapping (possible, but uncommon)

Case 1: pure on-line mapping
Resources are assigned (i.e. chosen) when a task is about to execute
- E.g. Assign to the processor where the task can be scheduled first; break ties by index
- On the previous example graph:
  - T₀ is mapped to PROC₀
  - T₁ is mapped to PROC₀ (tie broken)
  - T₂ is mapped to PROC₁ (starts earlier)
Mapping Techniques

**Case 2: pure off-line mapping**
Resources used by each task are completely specified assigned prior to the execution
- This is referred to as static mapping or partitioned scheduling as well
- The run-time support only has to implement the off-line choices

![Diagram showing resource allocation]

**Mapping Techniques**

**On-line vs off-line mapping**

↑ **On-line:** can adapt to dynamic scenarios.
  - unexpected tasks, variations in task duration...
↑ **On-line:** may provide higher performance in some cases.
  - this could may happen in periodic applications, since a task is allowed to be mapped to different resources in different iterations
↓ **On-line:** provides lower performance in most cases.
  - the mapping decisions must be fast → weaker optimization
↓ **On-line:** provides poor predictability.
↓ **On-line:** requires a more complex run-time support.
  - Decision making capabilities
  - Monitoring capabilities
Mapping Techniques

On-line vs off-line mapping

↓ **Off-line**: poor adaptability to changes.

↓ **Off-line**: may provide poorer performance in some cases.  
   *In a periodic application, the sequence of off-line task-to-resource mapping is necessarily finite*

↑ **Off-line**: provides higher performance in most cases.  
   *Mapping decisions can be strongly optimized prior to execution*

↑ **Off-line**: provides high predictability.

↑ **Off-line**: requires a very simple run-time support/hardware.

---

Off-line Optimization

**Solution Methods for Mapping & Scheduling**
Defining the Problem

For an off-line mapping & scheduling problem to be defined, we must have specified:
1. the application and platform description
2. the cost metrics and the constraints
3. the chosen run-time scheduling & mapping techniques

The actual resulting optimization problem depends on such choices
- “1” defines the problem input
- “2” defines the problem input
- “3” defines the problem output
- We have not yet considered “2” (we’ll see some examples in the followings)

Example 1: Classical Real Time OS

Model:
- Application description: set of tasks, periodic behavior
- Platform description: set of unary resources
- Cost Metrics: none
- Constraints: each task repetition must end by its deadline

Run-time support:
- Scheduling technique: fixed priority scheduling
- Mapping technique: pure on-line mapping

Resulting Problem:
Off-line choose task priorities so that no deadline can be violated at run-time
Example 2: Video Decoder on MPSoC

Model:
- Application description: an SDFG
- Platform description: set of unary resources (modeling processors)
- Cost Metrics: throughput (e.g. Frame rate of a video decoder)
- Constraints: none

Run-time support:
- Scheduling technique: fixed order on each processor
- Mapping technique: static mapping

Resulting Problem:
Off-line choose task-to-processor mapping and task ordering on each processor, in order to minimize throughput

Example 3: Simple Program on MPSoC

Model:
- Application description: a Task Graph
- Platform description: set of unary resources (modeling processors)
- Cost Metrics: completion time
- Constraints: hard deadline (maximum allowed completion time)

Run-time support:
- Scheduling technique: static scheduling
- Mapping technique: static mapping

Resulting Problem:
Off-line choose task-to-processor mapping and start times, in order to meet the deadline and minimize the completion time
Optimal vs Heuristic Approaches

In the following, we will focus on problem 3:

**Example Problem 3:**
Off-line choose task-to-processor mapping and start times, in order to meet the deadline and minimize the completion time

Solution methods for this problem can be classified into:
- **Optimal approaches** – able to find a guaranteed optimal solution
- **Heuristics approaches** – not able to provide any guarantee

The reason for using heuristic algorithms is that both computing an optimal mapping and an optimal schedule is NP-hard

---

Optimal vs Heuristic Approaches

Some recall on complexity classes:
- **$P$** = problems which can be solved in polynomial time (i.e. solution time $O(n^k)$)
- **NP-complete** = best algorithms known have exponential complexity (i.e. solution time $O(k^n)$)
- **NP-hard** = at least as hard as NP-complete

A polynomial time heuristics can provide quickly good solutions for large problems, where finding the optimum would take **centuries.**
A Heuristic Approach: List Scheduling

- Pretty old scheduling method from the Operation Research background [Kelley 1963, Kolish 1996]
- Very simple (and easy to implement)
- And yet very effective!
- Also known as (serial) Priority Based Scheduling

Basic idea:
- Map & schedule Tasks one by one, once their predecessors are over
- Select the task according to a priority rule
- Select the resources according to a priority rule

List Scheduling: an example

- Task priority:
  \[ pr(T_i) > pr(T_j) \iff i < j \]
- Resource priority:
  \[ pr(R_i) > pr(R_j) \iff \max(\text{start}(T' \text{ on } R_i)) < \max(\text{start}(T' \text{ on } R_j)) \]
**List Scheduling: Formal Description**

**Some notation:**
- $G = <T,A>$: the Task Graph to be scheduled
- $S$: set scheduled tasks

**List Scheduling Scheme**

```plaintext
while $S \neq T$:
    $D = \{t_i \in T \mid$ all predecessors in $G$ are in $S\}$
    $i^* = \text{argmax}_D(\text{pr}(t_i))$
    $\text{RES}^* = \{r^* \text{ for each class } | \ r^* = \text{argmax}_R(\text{pr}(R_r))\}$
    assign $t_i$, to resources in $\text{RES}^*$
    $\text{start}(t_i^*) = \min \{t \mid \text{all } R_r \text{ in } \text{RES} \text{ suff. capacity}\}$
    $S = S + t_i^*$.
```

**Suboptimality/Incompleteness Issues**

**The approach is heuristic**

- **non optimal**: The provided solution is not necessarily optimal
- **non complete**: If there are other constraints, a feasible solution is not guaranteed to be found, even if it exists

**Example**

- With the following deadline and the previous priorities, no feasible schedule is found

```
+---+---+---+---+---+---+---+---+---+---+
| P0 | T0 | T1 | T3 | T5 | T6 |
+---+---+---+---+---+---+---+---+---+---+
| P1 | T2 | T4 |
+---+---+---+---+---+---+---+---+---+---+
```

- But by swapping $T_4$ and $T_5$, everything would work!
LS: It’s all up to the Priorities

- Task priority:
  \[ \text{pr}(T_i) > \text{pr}(T_j) \iff \begin{cases} \text{start}(T_i) < \text{start}(T_j) \land || \\ (\text{start}(T_i) = \text{start}(T_j) \land i < j) \end{cases} \]

- Resource priority:
  \[ \text{pr}(R_i) > \text{pr}(R_j) \iff \max(\text{start}(T_i \text{ on } R_i)) < \max(\text{start}(T_i \text{ on } R_j)) \]

List Scheduling: Some Properties

**Dynamic/static priorities**

- Static: computed once for all at the beginning of the process
- Dynamic: re-computed during while the algorithm runs

**Note that:**

- Here we are talking to LS as an off-line algorithm
- LS is greedy: the first schedule produced is the only one produced

**An interesting theorem:**

- There exists always a priority assignments such that LS produces the optimal schedule :-)
- However, finding such priority assignment is as hard as solving the problem to optimality :-(
**Contradicting Common Wisdom**

**A typical (wrong!) reasoning**
Optimal mapping & schedule is NP-hard → exponential complexity
I can’t afford spending centuries → do not use optimal approaches

**Sol. Time over #tasks**
- What’s the polynomial curve?
- Hey! What happens here?

All depends on the size of my problem!

---

**An Optimal Approach: LP**

**Linear Programming**
- Optimization method devised within Operations Research in 1947
- well known to lots of researchers in other fields as well
- Declarative approach: once a problem is modeled via LP, it can be solved by off-the-shelf solvers → no algorithm to code!

**Linear Programming Model**, basic ingredients:
- Real, non-negative variables
- Linear constraints (equalities and/or inequalities)
- Linear objective function

**Solution method** (not detailed):
- Simplex algorithm (pseudo-polynomial)
- Interior point method (polynomial)
Mixed Integer Linear Programming

Unfortunately, a pure Linear Programming model is not sufficient to capture the problems we are interested in.

**Mixed Integer Linear Programming**

- same as Linear programming
- But some variables are constrained to be integer

**Solution method** (not detailed):

- Branch and bound, Branch and cut
- Exponential worst case solution time
- But many very good solvers are available off-the-shelf (e.g. IBM-ILOG Cplex)

---

MILP example

**Example Problem 3:**
Off-line choose task-to-processor mapping and start times, in order to meet the deadline and minimize the completion time

**Some notation:**

- \( G = <T, A> \): the Task Graph to be scheduled
- \( T^*(T) \): set of predecessors of \( T_i \)
- \( D_i \): constant duration of \( T_i \)
- \( C_{ri} \): amount of resource \( R_r \) required by \( T_i \)
- \( Cap_r \): capacity of resource \( R_r \)

For sake of simplicity: a single class of alternative resources
MILP example

Step 1: define variables
- \( S_{ik} \in \{0,1\} : S_{ik} = 1 \) iff \( T_i \) starts at time \( k \)
- \( U_{ir} \in \{0,1\} : U_{ir} = 1 \) iff \( T_i \) uses resource \( R_r \)
Span for the “\( k \)” index: 0..dline

Step 2: define constraints
This is all math (in practice, an MILP modeling language is used)
- Each task must start: \( \sum_{k=0}^{dline} S_{ik} = 1 \)
- Every task must end by the deadline: \( \sum_{k=0}^{dline} k \cdot S_{ik} + D_i \leq dline \)
- Prece. constraint \((T_{i},T_{j}) \in A:\) \( \sum_{k=0}^{dline} k \cdot S_{ik} + D_i \leq \sum_{k=0}^{dline} k \cdot S_{jk} \)

MILP example

Step 2: define constraints
- Each task must use a resource: \( \sum_{r=0}^{R} U_{ir} = 1 \)
- The capacity of the used resource must not be exceeded, then forall \( k = 0 \) to deadline:
  \[ \sum_{r} U_{ir} = 1 \sum_{h=k-d_i+1}^{k} C_{ri} S_{ih} \leq Cap_r \]
In practice:
- The sum of the requiremens of the task using the resource (sum of \( C_{ri} \) with \( U_{ir} = 1 \))
- Active at time \( k \) (hence, starting between “\( k-d_i+1 \)” and “\( k \)”)
- Must not exceed the resource capacity \( Cap \)
WARNING: this is not yet a linear constraint \( \to \) should be linearized
MILP example

Step 3: define the objective function
- For the problem at hand, the makespan is the end time of T₆:

\[
\min \sum_{k=0}^{d_{\text{line}}} k \cdot S_{6,k}
\]

Step 4: solve
The solver returns an assignment of all problem variables such that:
- All the constraints are satisfied
- The objective function is minimized
→ the optimal schedule

Optimal vs heuristics

Computed completion tim over deadline value

- Feasibility boundary
- Heuristics
- Optimal
- A heuristics may find suboptimal solutions
- When the deadline is tight, heuristics may find no solution at all
- But optimal approaches structurally have scalability problems
A case study

Scheduling & Voltage Scaling

Different voltages: different frequencies

Energy/speed trade-offs: varying the voltages

Voltage and Frequency scaling make the problem even harder!

Current off-line approaches solve mapping, scheduling and voltage selection separately (sequentially)

Power

Slack

Mapping and scheduling: given (fastest freq.)
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

Master problem (MP)
- Resource constraints
- SP Relaxation Constraint
  (eg. sum of ExecT on a Proc does not exceed deadline)
- Timing constraint

Subproblem (SP)
- Allocation & Freq. Assign.: INTEGER PROGRAMMING
- Scheduling: CONSTRAINT PROGRAMMING

All+Sch UB for cost
Allocation LB for cost

Obj. Function
eg. system energy
In general, it depends on MP & SP variables

No good: unfeasible SP
Optimality cut: SP solution is optimal UNLESS a better one exists with a different allocation

Iterations stop when MP becomes unfeasible!
Computational scalability

Deterministic task graphs, mapping & scheduling

- 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

Stochastic task graphs, mapping & scheduling & min bus usage

- Hundreds of decision variables
- Much beyond ILP solver or CP solver capability
Does it Matter?

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>
<tr>
<td>- 66.4%</td>
<td></td>
</tr>
</tbody>
</table>

Optimality gap

Comparison with heuristic 2-phase solution (GA)

“timing barrier”

Gap significant when constraints are tight
Final Remarks
Putting it all together, an making a tool

MPOpt

A collection of design tools to assist software development over parallel hardware platforms

A threefold (long-term) objective:
- Ease code development/parallelization
- Provide high performance
- Code retargeting support

Currently a development prototype of the full toolchain is available for CELL BE → MPOpt-CELL
- Hardware resources
- Code retargeting support
MPOpt-CELL

Optimized Compiler Flow:

- **Entry point**: OpenMP code with custom annotation, defining a Synchronous Dataflow Graph
- **Exit point**: instrumented code, to be compiled with the IBM SDK

---

MPOpt-CELL

- Extract kernel code
- Extract a Dataflow Graph description of the application (Synchronous Dataflow Graph)
- Compute an optimized, partially static mapping & schedule
- Add code for:
  - Kernel synchronization (a single coordinator thread on the PPE)
  - DMA data transfers (double buffering to hide transfer time)
Comparison with CellSs

A comparison with a dynamic mapping & scheduling approach

http://mpopt.ing.unibo.it

That’s it!
...Questions?