Energy Minimisation for Parallel Fork-Join Tasks on an Embedded Real-Time Operating System

Antonio Paolillo - Ph.D student & software engineer

GT OVSTR 2nd conference on Optimizing Real-Time Systems
Theme: parallel real-time tasks

23th May 2016
Energy Minimisation for Parallel Fork-Join Tasks on an Embedded Real-Time Operating System
Energy Minimisation for Parallel Fork-Join Tasks on an *Embedded* Real-Time Operating System
Energy Minimisation for Parallel Fork-Join Tasks on an Embedded Real-Time Operating System
Energy Minimisation for Parallel Fork-Join Tasks on an Embedded Real-Time Operating System
Energy Minimisation for **Parallel Fork-Join Tasks**
on an Embedded Real-Time Operating System
Energy Minimisation for Parallel Fork-Join Tasks on an Embedded Real-Time Operating System
Energy Minimisation for Parallel Fork-Join Tasks on an Embedded **Real-Time Operating System**
Sequential schedule
Sequential schedule
Sequential schedule
Sequential schedule
Sequential schedule
Sequential schedule
Sequential schedule
Sequential schedule
Sequential schedule
Sequential schedule
Sequential schedule
Sequential schedule
Sequential schedule
Sequential schedule
Goal = Save power
Power-aware schedule
Power-aware schedule
Power-aware schedule
Power-aware schedule
Power-aware schedule
Power-aware schedule
Power-aware schedule
“The minimum speed is limited by the sequential job model”

- J. Anderson, S. Baruah
Parallel schedule
Parallel schedule
Parallel schedule
Parallel schedule
Parallel schedule
Parallel schedule
Parallel schedule
Parallel schedule
Parallel schedule
Parallel schedule
Parallel schedule
Parallel schedule
Parallel schedule
Can we save more with parallelism?
Parallel schedule
Save more with parallelism? Yes!

Combine:

- parallel task model
- Power-aware multi-core scheduling

→ validated in **theory** → **malleable** jobs (RTCSA’14)
This work

Goal:

Reduce as much possible **energy** while meeting **real-time requirements**.

Validate it *in practice*

How? Combine:

- Parallel programming model
- RTOS techniques → power-aware multi-core scheduling
Contributions

- Energy-efficient parallel run-time framework for hard real-time applications
- Basic scheduling analysis
- Evaluated with full experimental stack (RTOS+hardware in the loop)
Talk agenda

A. Present the run-time framework

B. Present the task model and the schedulability analysis

C. Present experiments and discuss results
A. Framework
Run-time framework
Run-time framework

- RTOS & lib support
- MPSoc platform
Run-time framework

RTOS & lib support

MPSoc platform
Run-time framework

- MPSoc platform
- RTOS & lib support
- Parallel app. benchmark
Run-time framework

- OpenMP
- Parallel app. benchmark
- RTOS & lib support
- MPSoc platform
Parallel programming model

OpenMP™
(Simple) Parallel Program

```c
int main()
{
    const int num_steps = 1000;
    omp_set_num_threads(4);

#pragma omp parallel
{
    int nbt = omp_get_num_threads();
    int tid = omp_get_thread_num();
    int i_start = (tid * num_steps) / nbt;
    int i_end = ((tid + 1) * num_steps) / nbt;

    for (int i = i_start; i < i_end; ++i) {
        /* workload executed in parallel */
    }
}

    return 0;
}
```
(Simple) Parallel Program Flow
(Simple) Parallel Program Flow

```c
int main()
{
    const int num_steps = 1000;
    omp_set_num_threads(4);
    #pragma omp parallel
    {
        int nbt = omp_get_num_threads();
        int tid = omp_get_thread_num();
        int i_start = (tid * num_steps) / nbt;
        int i_end = ((tid + 1) * num_steps) / nbt;
        for (int i = i_start; i < i_end; ++i) {
            /* workload executed in parallel */
        }
    }
    return 0;
}
```
An embedded RTOS
An embedded RTOS

- Micro-kernel based
- Natively supports multi-core
- Provides hard-real time scheduling
- Power management fixed at task level

We developed HOMPRTL

→ OpenMP run-time support for HIPPEROS.
B. Analysis
Parallel task model
Parallel task model
Parallel task model

- Periodic fork-join
- Implicit deadlines
- Rate monotonic based analysis
- Threads are partitioned
- 3 stages, very simple
Platform model

- Symmetric Multi-Core
- Global DVFS, finite set of operating points $< V_{dd}, f >$
- Power increases monotonically with frequency
Optimisation & partitioning process
Optimisation & partitioning process

- Power optimisation (for each degree of parallelism)
  - Static assignment of opp to tasks
  - Optimisation heuristic
  - WCETs are known for each opp / degree of parallelism
Optimisation & partitioning process

- **Power optimisation** (for each degree of parallelism)
  - Static assignment of opp to tasks
  - Optimisation heuristic
  - WCETs are known for each opp / degree of parallelism

- **Partitioner**
  - Search algorithm with constraints
  - Branch and bound exploration
  - $O(2^n)$
Optimisation & partitioning process

● Power optimisation (for each degree of parallelism)
  ○ Static assignment of opp to tasks
  ○ Optimisation heuristic
  ○ WCETs are known for each opp / degree of parallelism

● Partitioner
  ○ Search algorithm with constraints
  ○ Branch and bound exploration
  ○ $O(2^n)$

● Partitioned schedulability test
  ○ Response-time bound of fork join, Axer et al (ECRTS’13)
  ○ Fixed an infinite loop problem
Optimisation & partitioning process

- **Power optimisation** (for each degree of parallelism)
  - Static assignment of opp to tasks
  - Optimisation heuristic
  - WCETs are known for each opp / degree of parallelism

- **Partitioner**
  - Search algorithm with constraints
  - Branch and bound exploration
  - \(O(2^n)\)

- **Partitioned schedulability test**
  - Response-time bound of fork join, Axer et al (ECRTS’13)
  - Fixed an infinite loop problem

- **Online scheduler**
  - Partitioned Rate Monotonic
  - Static speed assignment
Optimisation & partitioning process

Input system

Output solution

Next partitioning

Setup power

Next power config

Setup partitioning

Scheduling test

Register solution

set exhausted

failure

success

87
C. Experiments
Run-time framework
Experimental stack

- MPSoc platform
- RTOS & lib support
- Parallel app. benchmark
- OpenMP
Experimental stack

- OpenMP
- Parallel app. benchmark
- RTOS & lib support
- MPSoC platform
- Measurement framework
Experimental stack

- Parallel app. benchmark
- RTOS & lib support
- MPSoC platform
- Measurement framework
Target platform - i.MX6q SabreLite

- Embedded board ARM Cortex A9 MP
- 4 cores, but global DVFS
- Operating points
- Supported by HIPPEROS
## Operating points

<table>
<thead>
<tr>
<th>Voltage (mV)</th>
<th>Frequency (MHz)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1100</td>
<td>876</td>
</tr>
<tr>
<td>1125</td>
<td>948</td>
</tr>
<tr>
<td>1150</td>
<td>1008</td>
</tr>
<tr>
<td>1175</td>
<td>1068</td>
</tr>
<tr>
<td>1200</td>
<td>1116</td>
</tr>
<tr>
<td>1225</td>
<td>1164</td>
</tr>
<tr>
<td>1250</td>
<td>1212</td>
</tr>
<tr>
<td>1275</td>
<td>1260</td>
</tr>
<tr>
<td>1300</td>
<td>1296</td>
</tr>
</tbody>
</table>
Measurement framework

- Keysight Agilent Mso9254a probe
- Voltage measurements
- TCP/IP communication
- Automated with Python → including offline tests
MPSoC
Embedded board
MPSoC

Embedded board

Power supply

220 V
MPSoC
Embedded board
Oscilloscope
Power supply
Transformer
220 V
5 V
220 V
Oscilloscope

Embedded board

MPSoC

Power supply

220 V

Transformer

5 V

R
Oscilloscope

Voltage measurements

\[ P = \frac{V^2}{R} \]

Embedded board

Power supply

Transformer

MPSoC

103
Voltage measurements

\[ P = \frac{V^2}{R} \]
Voltage measurements

\[ P = \frac{V^2}{R} \]
Voltage measurements
\[ P = V^2 / R \]
Voltage measurements

\[ P = V^2 / R \]
Voltage measurements:

\[ P = \frac{V^2}{R} \]
The diagram shows a circuit setup involving a Control Computer, Oscilloscope, Embedded board, Power supply, and various connections like USB, Serial, JTAG, and HIPPEROS Inputs Outputs. The equation presented is \( P = \frac{V^2}{R} \) for voltage measurements.
Control Computer

HIPPEROS
Inputs Outputs

MPSoC
Embedded board

Oscilloscope

USB

Serial
JTAG

Power supply

220 V

Transformer

5 V

220 V

Voltage measurements

P = V^2 / R
Control Computer

MPSoC

Embedded board

Oscilloscope

HIPPEROS Inputs Outputs

Serial

JTAG

USB

P = V^2 / R

Voltage measurements

Transformer

Power supply

5 V

220 V

220 V
Control Computer

Captured data transfer

Oscilloscope

Captured data transfer

HIPPEROS
Inputs Outputs

Serial

JTAG

Embedded board

MPSoC

Oscilloscope

Voltage measurements

$P = \frac{V^2}{R}$

Transformer

5 V

220 V

Power supply

220 V

USB
Voltage trace
Voltage trace
Use cases

- OpenMP
- Parallel app. benchmark
- RTOS & lib support
- MPSoc platform
- Measurement framework
Use cases

- MPSoc platform
- RTOS & lib support
- Parallel app. benchmark
- Measurement framework
- Pi
- Image
- AES
Use cases

- Different workloads
- Easy OpenMP implementation
- Good scaling expected
Use case alone - voltage probe output
Use case alone - execution time
Use case alone - peak power
Use case alone - energy
Experiments
Experiments

- Generate 350 random task systems $(U = 0.6 \rightarrow 4.0)$
Experiments

- Generate 350 random task systems \((U = 0.6 \rightarrow 4.0)\)
- Bind generated task to use cases
Experiments

- Generate 350 random task systems ($U = 0.6 \rightarrow 4.0$)
- Bind generated task to use cases
- Scheduling test & optimisation in Python for 1, 2, 3, 4 threads
Experiments

- Generate 350 random task systems ($U = 0.6 \rightarrow 4.0$)
- Bind generated task to use cases
- Scheduling test & optimisation in Python for 1, 2, 3, 4 threads
- Operating points and threads partition for each task
Experiments

- Generate 350 random task systems \((U = 0.6 \rightarrow 4.0)\)
- Bind generated task to use cases
- Scheduling test & optimisation in Python for 1, 2, 3, 4 threads
- Operating points and threads partition for each task
- Generate HIPPEROS builds
Experiments

● Generate 350 random task systems \((U = 0.6 \rightarrow 4.0)\)

● Bind generated task to use cases

● Scheduling test & optimisation in Python for 1, 2, 3, 4 threads

● Operating points and threads partition for each task

● Generate HIPPEROS builds

● Run feasible systems and measures
Schedulability ratio

- 350 systems
  295 at least one // solution
  198 for all //

- Up is good

- No high U system (P-RM)

- Pessimistic with more threads
Energy consumption

- All degree of //ism only
- Down is good
- Low U: don’t need high OPP
  Often idle
- More threads tends to consume less
Relative energy savings

- Same systems, but rel. to 1 thread
- //ism helps to save E :-)  
- Stdev < 0.13
- “4 threads” dominates
- Increase //ism improves savings
  Less for high U: - pessimism
  - even repartition
Conclusions

- Practical experimental framework flow for parallel real-time applications
- Benefits of parallelisation: energy savings up to 25%
Conclusions

● Practical experimental framework flow for parallel real-time applications

● Benefits of parallelisation: energy savings up to 25%

● Future work
  ○ Realistic use cases, other parallel models
  ○ Advanced scheduling techniques (global, EDF, etc.)
  ○ Better analyses
  ○ DVFS + switch on/off cores
  ○ Other metrics (e.g. T°)
  ○ Other platforms (more determinism?)
Thank you! Q?