A runtime DVFS algorithm beyond communication slack

Amina Guermouche

February 6, 2015
Introduction

• Computational power
  Tianhe-2 33,863 TFlops

• Increase in the number of components
  Tianhe-2 3,120,000 cores

• Large power consumption on powerful supercomputers
  Tianhe-2 17MW
  (Chad power consumption: \(\sim 10MW\))
Dynamic voltage and frequency scaling

Energy = Power \times Execution\_Time

Power = f(\text{frequency}^3)

- Dynamic Voltage and Frequency scaling (DVFS)
  - Reduce frequency:
    - Reduce power consumption
    - Reduce energy consumption
      if the slowdown due to lower frequency does not dramatically increase execution time

![Graph showing time and energy consumption](image)

**Figure**: Time and energy consumption of an application on a Nehalem 2 \times 6 cores.
Existing DVFS solutions

1. Offline scheduling
   - Full knowledge of application behavior
   - Best frequency (frequencies) execution of each phase of the application

2. Runtime systems
   - Iterative deterministic applications
   - Best frequency schedule according to the profiling
Outline

1. Basic principles
2. FoREST-mn
3. Experiments results
Basic principles

- MPI applications
- Task graph
- Dynamic Voltage and Frequency Scaling (DVFS)

```c
if (rank == 0) {
    ... (T1)
    MPI_Send(1, ...)
    ... (T2)
    MPI_Recv(1, ...)
    ... (T3)
} else {
    ... (T4)
    MPI_Recv(0, ...)
    ... (T5)
    MPI_Send(0, ...)
    ... (T6)
}
```
Frequency impact on execution time

- Longer execution time
- Communication slack
  - slack: time spent by a task awaiting a message

![Diagram showing processor time and tasks]

- Computation
- Communication
- Slack
- Task slowdown
- Message
Existing solution: Adagio

- Slow tasks down to reduce the slack while enforcing the deadline
  - Lower frequency
    - Reduce power consumption
  - Fixed execution time
    - Reduce energy consumption

☹ Works only on unbalanced codes
Outline

1. Basic principles
2. FoREST-mn
3. Experiments results
Forest-mn basic principles

- Iterative and deterministic applications

1. Profiling:
   - Generate the task graph
   - Profile the tasks (their behavior according to frequency)

2. Running the first iteration
   - Compute and apply the **locally optimal frequency**
     - Choose the frequency providing the best energy gain for each task
     - Regardless its effects on the other tasks
Apply the optimal local frequency

![Diagram showing processor scheduling]

- Time
- Processor
- Computation
- Communication
- Slack
- Message

- Additional slack generation
- May lead to more energy consumption

How to remove the additional slack?
Apply the optimal local frequency

- Additional slack generation
  ⇒ May lead to more energy consumption

How to remove the additional slack?
How to remove additional slack?

1\textsuperscript{st} possibility:

- Slow down tasks preceding the slack
- A lot of tasks have to be slowed down

\( \Rightarrow \) energy consumption

Recall:

Locally optimal frequency

\( \Rightarrow \) Lowest energy consumption
How to remove additional slack?

2nd possibility:
- Speed up tasks generating slack
- Limited impact on energy
How to remove additional slack?

2nd possibility:
- Speed up tasks generating slack
  - Limited impact on energy
  - Tasks with slack:
    ⇒ Send a message to the original sender
- Overhead
- Paid only once in the program execution
Globally optimal frequency

![Diagram showing computation, communication, and slack times for different processors and time steps.](image-url)
Drive slow

- Globally optimal frequency
  - Generates some slowdown
  - User might have deadlines

⇒ Slowdown constraints
  - User-defined
  - Using speedup profiling to limit the range of allowed frequencies
    (5%, 10%, 20%, 100%)
Outline

1. Basic principles
2. FoREST-mn
3. Experiments results
Forest-mn basic architecture

- Hooked between the program executable and the MPI library
- Uses the PMPI interface provided in OpenMPI
Frequency transition:
- Group tasks as long as group execution time is less than 50 times the transition time
- Execute group at the maximal frequency requested by all the tasks in that group

Multicore:
- Add additional synchronization at the loop boundary during the profiling phase and before the optimal iteration
- Cores cannot ask for another frequency as long as other cores have not reached the end of the loop
- More accurate profiling at the processor level
Experimental setup

Intel RAPL:
- CPU energy consumption

Benchmarks:
- Class D NAS Benchmark 3.3.1 on 64 processes
- 5 runs and the median is used

Experimental setup:
- 4 nodes
- 2 Intel Xeon E5-2670 CPU per node
- 8 cores per processor
- 64GB of RAM
Experiments details

- **ondemand**
  - Default linux DVFS controller
  - Intra-node DVFS
  - Increase frequency proportionally to CPU Load

- **Adagio**
  - Commercial inter-node DVFS controller
  - Reduce task frequency to reduce slack
    (not publicly available, we had to implement it)
CPU energy consumption

**Figure:** Energy consumption of FoREST-mn with slowdown constraints varying from 5% to 20%, compared to Adagio, expressed relatively to ondemand.
**Execution time comparison**

**Figure:** Execution time of FoREST-mn with slowdown constraints varying from 5% to 20%, relatively to the maximal frequency fmax.
System energy consumption

1. Yokogawa WT200:
   - System energy consumption

2. Benchmarks:
   - Class C NAS Benchmark 3.3.1 on 8 processes

3. Setup:
   - 1 node
   - 1 Intel Core i7 3770 CPU
   - 4 cores and 2 threads per core
   - 16 GB of RAM
## CPU and system energy consumption

<table>
<thead>
<tr>
<th>Program</th>
<th>Execution time</th>
<th>CPU energy</th>
<th>System energy</th>
</tr>
</thead>
<tbody>
<tr>
<td>BT</td>
<td>1.09</td>
<td>0.84</td>
<td>0.93</td>
</tr>
<tr>
<td>CG</td>
<td>1.05</td>
<td>0.85</td>
<td>0.93</td>
</tr>
<tr>
<td>FT</td>
<td>1.03</td>
<td>0.97</td>
<td>0.99</td>
</tr>
<tr>
<td>IS</td>
<td>1.13</td>
<td>0.90</td>
<td>0.99</td>
</tr>
<tr>
<td>LU</td>
<td>1.01</td>
<td>0.96</td>
<td>0.98</td>
</tr>
<tr>
<td>MG</td>
<td>1.02</td>
<td>0.60</td>
<td>0.77</td>
</tr>
<tr>
<td>SP</td>
<td>1.00</td>
<td>0.92</td>
<td>0.95</td>
</tr>
</tbody>
</table>

*Table:* Execution time, CPU energy, and whole-system energy consumption of a single computer with FoREST-mn allowed to perform at most 10% slowdowns.
Outline

1. Basic principles
2. FoREST-mn
3. Experiments results
Conclusion

1. **Conclusion**
   - Energy savings even on well balanced codes
   - Customizable user-defined slowdown constraint

2. **Parallel work:**
   - Offline scheduling to provide optimal energy consumption
   - NP-Hard problem when considering architecture constraints