Energy-Aware real-time task partitioning

Parallel or Sequential?

Houssam-Eddine ZAHAF
houssam.zahaf@unimore.it

HiPeRT Lab, University of Modena and Reggio Emillia

OSPM, Pisa, 18-20/03/2018

Plan

Parallelization in real-time systems

Time and energy models

From analysis to implementation

Conclusions and future work

Plan

Parallelization in real-time systems

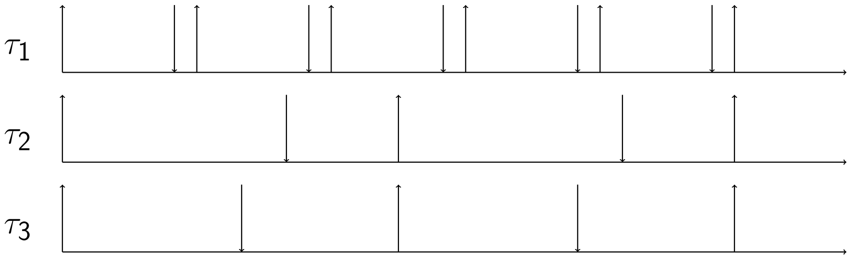
Time and energy models

From analysis to implementation

Conclusions and future work

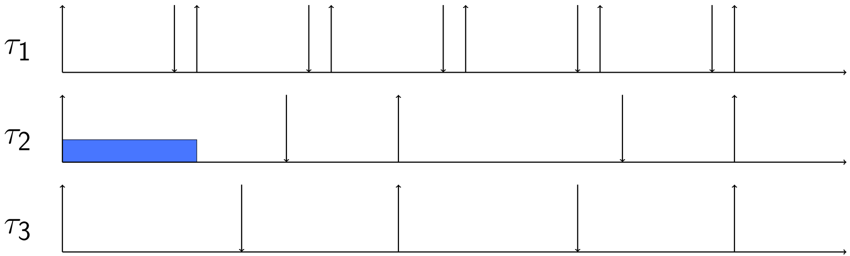
Introduction: partitionned scheduling, parallelization

- Let assume 3 tasks to allocate onto 2 cores.
- Scheduling policy: Earliest Deadline First.
- Allocation: $\tau_1, \tau_3 \Rightarrow$ Yellow core, $\tau_2 \Rightarrow$ Blue core



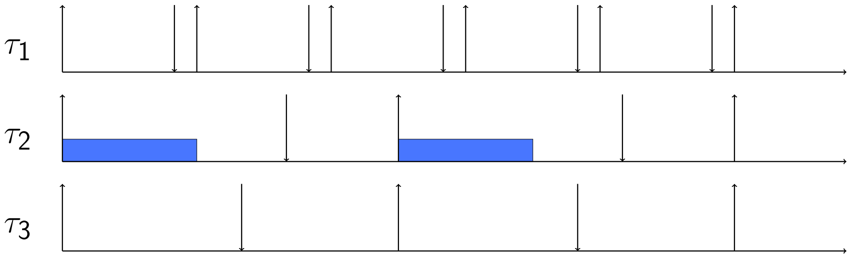
Introduction: partitionned scheduling, parallelization

- Let assume 3 tasks to allocate onto 2 cores.
- Scheduling policy: Earliest Deadline First.
- Allocation: $\tau_1, \tau_3 \Rightarrow$ Yellow core, $\tau_2 \Rightarrow$ Blue core



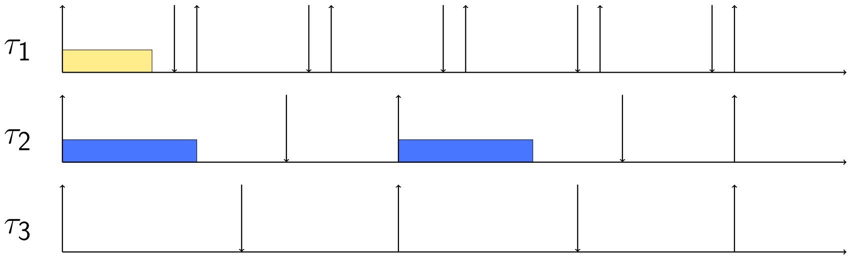
Introduction: partitionned scheduling, parallelization

- Let assume 3 tasks to allocate onto 2 cores.
- Scheduling policy: Earliest Deadline First.
- Allocation: $\tau_1, \tau_3 \Rightarrow$ Yellow core, $\tau_2 \Rightarrow$ Blue core



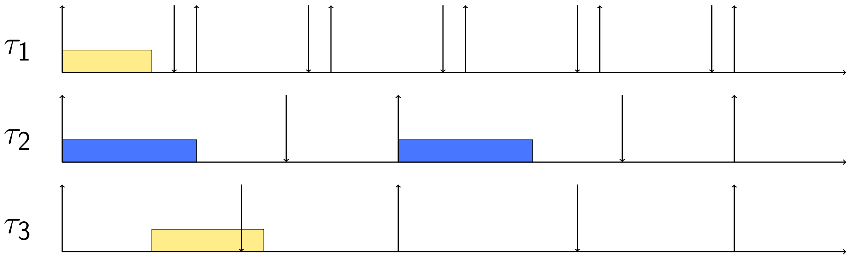
Introduction: partitionned scheduling, parallelization

- Let assume 3 tasks to allocate onto 2 cores.
- Scheduling policy: Earliest Deadline First.
- Allocation: $\tau_1, \tau_3 \Rightarrow$ Yellow core, $\tau_2 \Rightarrow$ Blue core



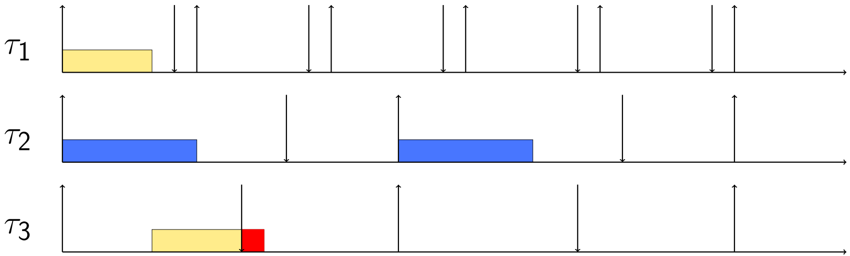
Introduction: partitionned scheduling, parallelization

- Let assume 3 tasks to allocate onto 2 cores.
- Scheduling policy: Earliest Deadline First.
- Allocation: $\tau_1, \tau_3 \Rightarrow$ Yellow core, $\tau_2 \Rightarrow$ Blue core



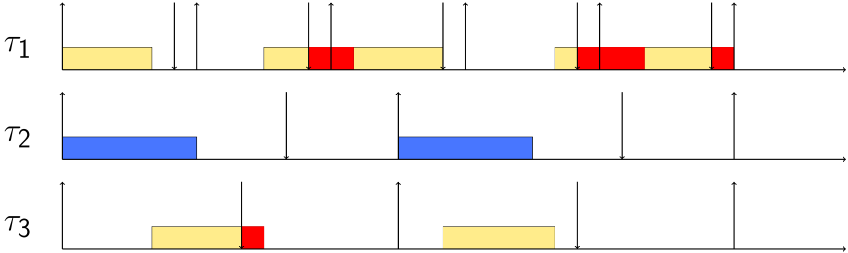
Introduction: partitionned scheduling, parallelization

- Let assume 3 tasks to allocate onto 2 cores.
- Scheduling policy: Earliest Deadline First.
- Allocation: $\tau_1, \tau_3 \Rightarrow$ Yellow core, $\tau_2 \Rightarrow$ Blue core



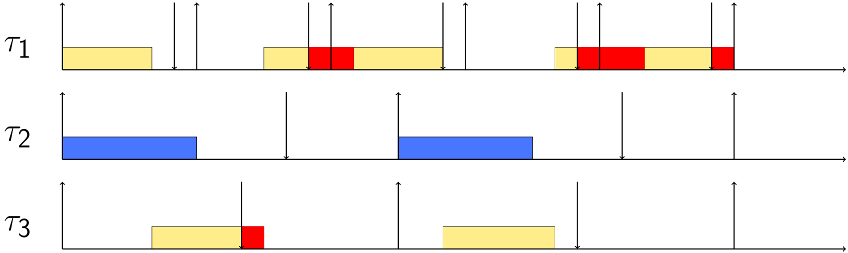
Introduction: partitionned scheduling, parallelization

- Let assume 3 tasks to allocate onto 2 cores.
- Scheduling policy: Earliest Deadline First.
- Allocation: $\tau_1, \tau_3 \Rightarrow$ Yellow core, $\tau_2 \Rightarrow$ Blue core



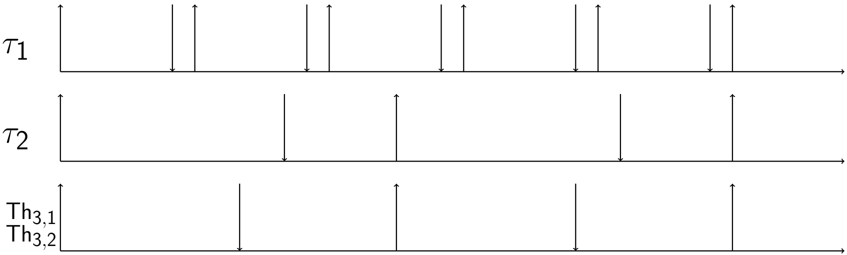
Introduction: partitionned scheduling, parallelization

- Let assume 3 tasks to allocate onto 2 cores.
- Scheduling policy: Earliest Deadline First.
- Allocation: $\tau_1, \tau_3 \Rightarrow$ Yellow core, $\tau_2 \Rightarrow$ Blue core
- "Allocate" 2 time units from τ_3 onto blue core



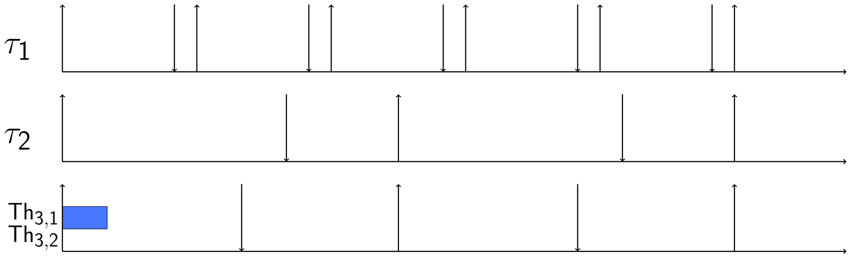
Introducton: partitioned parallel EDF

- Let assume 3 tasks to allocate onto 2 cores.
- Scheduling policy: Earliest Deadline First.
- Allocation: $\tau_1, \tau_3 \Rightarrow$ Yellow core, $\tau_2 \Rightarrow$ Blue core
- "Allocate" 2 time units from τ_3 onto blue core



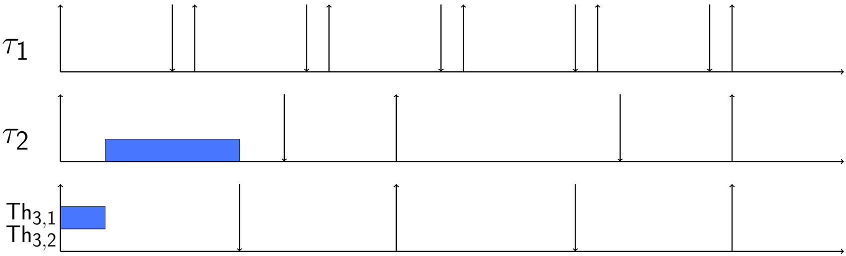
Introducton: partitioned parallel EDF

- Let assume 3 tasks to allocate onto 2 cores.
- Scheduling policy: Earliest Deadline First.
- Allocation: $\tau_1, \tau_3 \Rightarrow$ Yellow core, $\tau_2 \Rightarrow$ Blue core
- "Allocate" 2 time units from τ_3 onto blue core



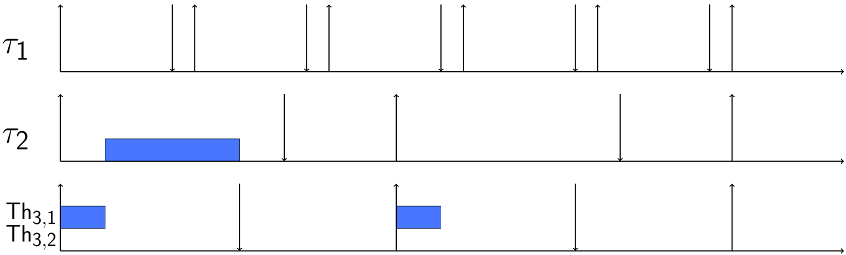
Introducton: partitioned parallel EDF

- Let assume 3 tasks to allocate onto 2 cores.
- Scheduling policy: Earliest Deadline First.
- Allocation: $\tau_1, \tau_3 \Rightarrow$ Yellow core, $\tau_2 \Rightarrow$ Blue core
- "Allocate" 2 time units from τ_3 onto blue core



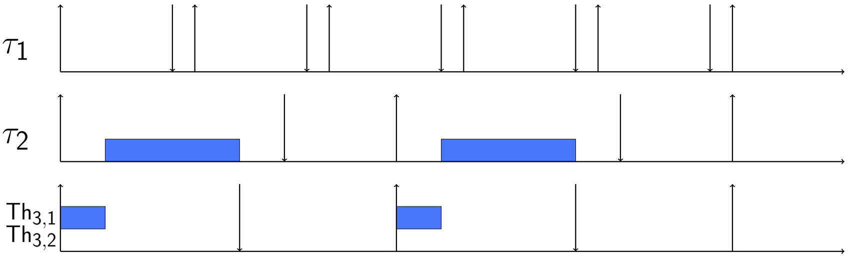
Introducton: partitioned parallel EDF

- Let assume 3 tasks to allocate onto 2 cores.
- Scheduling policy: Earliest Deadline First.
- Allocation: $\tau_1, \tau_3 \Rightarrow$ Yellow core, $\tau_2 \Rightarrow$ Blue core
- "Allocate" 2 time units from τ_3 onto blue core



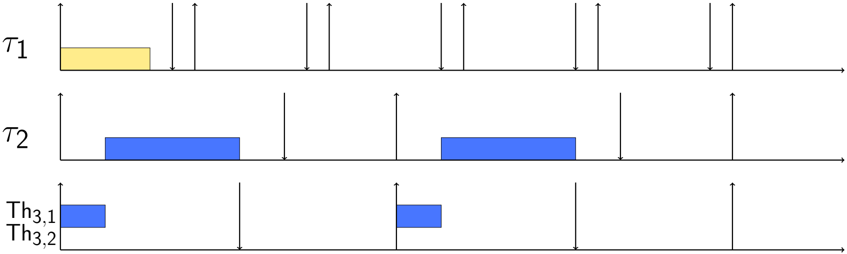
Introducton: partitioned parallel EDF

- Let assume 3 tasks to allocate onto 2 cores.
- Scheduling policy: Earliest Deadline First.
- Allocation: $\tau_1, \tau_3 \Rightarrow$ Yellow core, $\tau_2 \Rightarrow$ Blue core
- "Allocate" 2 time units from τ_3 onto blue core



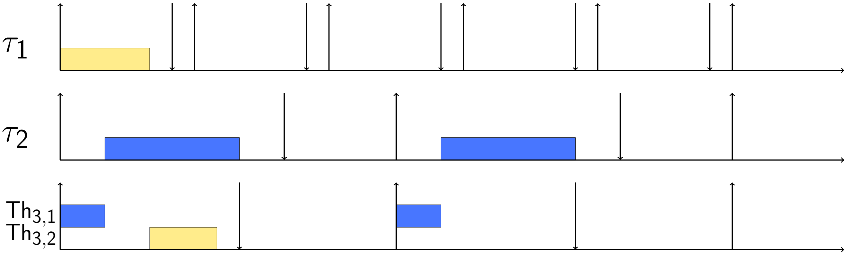
Introducton: partitioned parallel EDF

- Let assume 3 tasks to allocate onto 2 cores.
- Scheduling policy: Earliest Deadline First.
- Allocation: $\tau_1, \tau_3 \Rightarrow$ Yellow core, $\tau_2 \Rightarrow$ Blue core
- “Allocate” 2 time units from τ_3 onto blue core



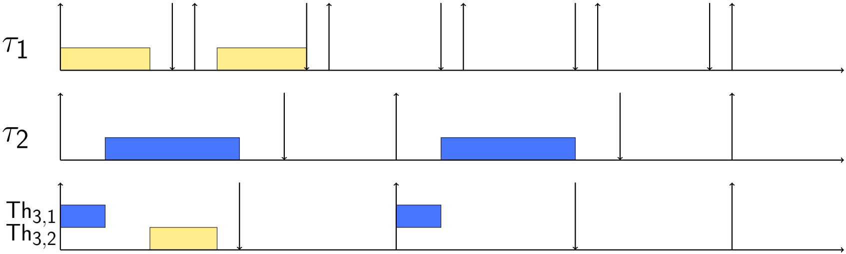
Introducton: partitioned parallel EDF

- Let assume 3 tasks to allocate onto 2 cores.
- Scheduling policy: Earliest Deadline First.
- Allocation: $\tau_1, \tau_3 \Rightarrow$ Yellow core, $\tau_2 \Rightarrow$ Blue core
- "Allocate" 2 time units from τ_3 onto blue core



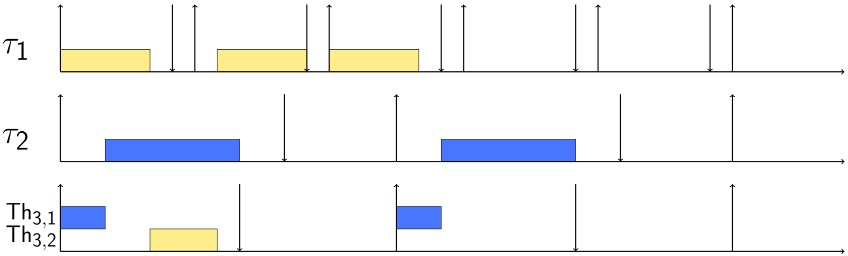
Introducton: partitioned parallel EDF

- Let assume 3 tasks to allocate onto 2 cores.
- Scheduling policy: Earliest Deadline First.
- Allocation: $\tau_1, \tau_3 \Rightarrow$ Yellow core, $\tau_2 \Rightarrow$ Blue core
- "Allocate" 2 time units from τ_3 onto blue core



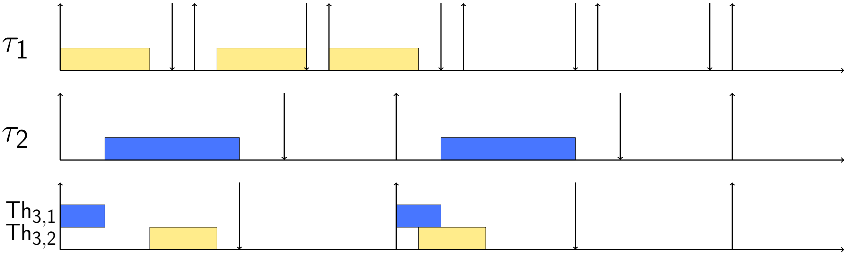
Introducton: partitioned parallel EDF

- Let assume 3 tasks to allocate onto 2 cores.
- Scheduling policy: Earliest Deadline First.
- Allocation: $\tau_1, \tau_3 \Rightarrow$ Yellow core, $\tau_2 \Rightarrow$ Blue core
- "Allocate" 2 time units from τ_3 onto blue core



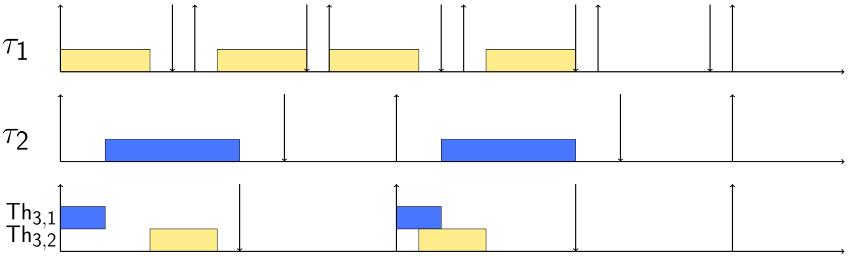
Introducton: partitioned parallel EDF

- Let assume 3 tasks to allocate onto 2 cores.
- Scheduling policy: Earliest Deadline First.
- Allocation: $\tau_1, \tau_3 \Rightarrow$ Yellow core, $\tau_2 \Rightarrow$ Blue core
- "Allocate" 2 time units from τ_3 onto blue core



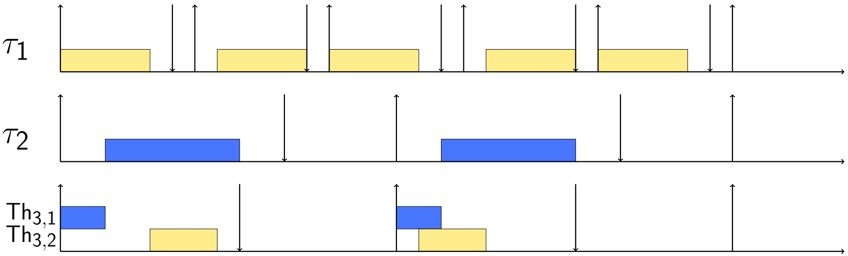
Introducton: partitioned parallel EDF

- Let assume 3 tasks to allocate onto 2 cores.
- Scheduling policy: Earliest Deadline First.
- Allocation: $\tau_1, \tau_3 \Rightarrow$ Yellow core, $\tau_2 \Rightarrow$ Blue core
- "Allocate" 2 time units from τ_3 onto blue core



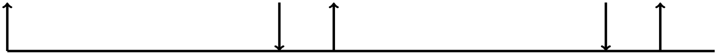
Introducton: partitioned parallel EDF

- Let assume 3 tasks to allocate onto 2 cores.
- Scheduling policy: Earliest Deadline First.
- Allocation: $\tau_1, \tau_3 \Rightarrow$ Yellow core, $\tau_2 \Rightarrow$ Blue core
- "Allocate" 2 time units from τ_3 onto blue core



Parallelization and frequency selection

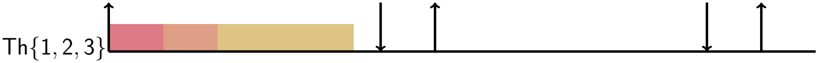
Let τ be a task that can be decomposed into 3 threads $\mathrm{Th}_1, \mathrm{Th}_2, \mathrm{Th}_3$, with the execution times 2,2,5 respectively on a platform operating at speed $s = 1$



Power $\propto$ Frequency^3

Parallelization and frequency selection

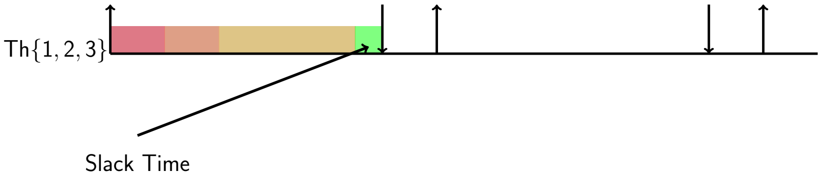
Let τ be a task that can be decomposed into 3 threads $\mathrm{Th}_1, \mathrm{Th}_2, \mathrm{Th}_3$, with the execution times 2,2,5 respectively on a platform operating at speed $s = 1$



$$\text{Power} \propto \text{Frequency}^3$$

Parallelization and frequency selection

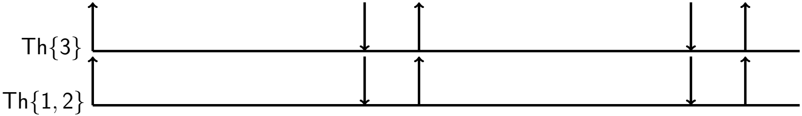
Let τ be a task that can be decomposed into 3 threads $\mathrm{Th}_1, \mathrm{Th}_2, \mathrm{Th}_3$, with the execution times 2,2,5 respectively on a platform operating at speed $s = 1$



$$\text{Power} \propto \text{Frequency}^3$$

Parallelization and frequency selection

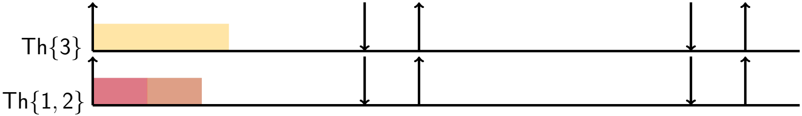
Let τ be a task that can be decomposed into 3 threads Th_1, Th_2, Th_3, with the execution times 2,2,5 respectively on a platform operating at speed $s = 1$



Power $\propto$ Frequency^3

Parallelization and frequency selection

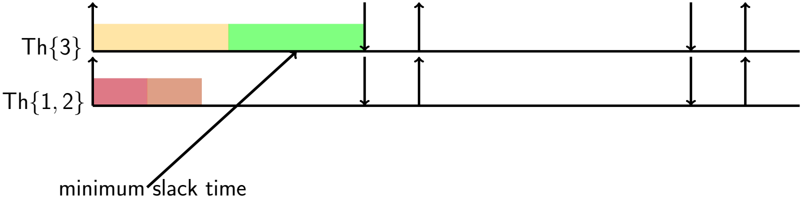
Let τ be a task that can be decomposed into 3 threads $\mathrm{Th}_1, \mathrm{Th}_2, \mathrm{Th}_3$, with the execution times 2,2,5 respectively on a platform operating at speed $s = 1$



Power $\propto$ Frequency^3

Parallelization and frequency selection

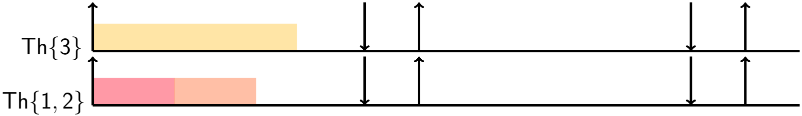
Let τ be a task that can be decomposed into 3 threads $\mathrm{Th}_1, \mathrm{Th}_2, \mathrm{Th}_3$, with the execution times 2,2,5 respectively on a platform operating at speed $s = 1$



Power $\propto$ Frequency^3

Parallelization and frequency selection

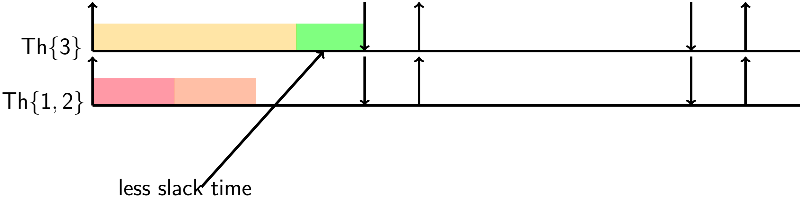
Let τ be a task that can be decomposed into 3 threads $\mathrm{Th}_1, \mathrm{Th}_2, \mathrm{Th}_3$, with the execution times 2,2,5 respectively on a platform operating at speed $s = 0.75$



Power $\propto$ Frequency^3

Parallelization and frequency selection

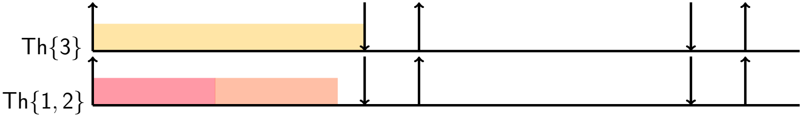
Let τ be a task that can be decomposed into 3 threads $\mathrm{Th}_1, \mathrm{Th}_2, \mathrm{Th}_3$, with the execution times 2,2,5 respectively on a platform operating at speed $s = 0.75$



$$\text{Power} \propto \text{Frequency}^3$$

Parallelization and frequency selection

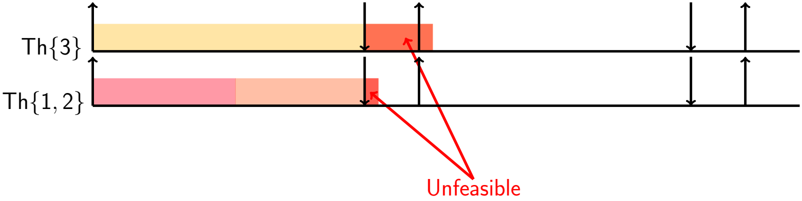
Let τ be a task that can be decomposed into 3 threads $\mathrm{Th}_1, \mathrm{Th}_2, \mathrm{Th}_3$, with the execution times 2,2,5 respectively on a platform operating at speed $s = 0.50$



Power $\propto$ Frequency^3

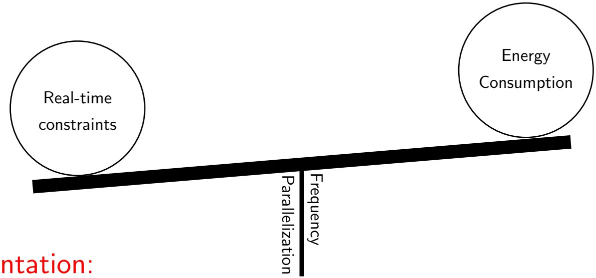
Parallelization and frequency selection

Let τ be a task that can be decomposed into 3 threads $\mathrm{Th}_1, \mathrm{Th}_2, \mathrm{Th}_3$, with the execution times 2,2,5 respectively on a platform operating at speed $s = 0.35$



Power $\propto$ Frequency^3

Problem



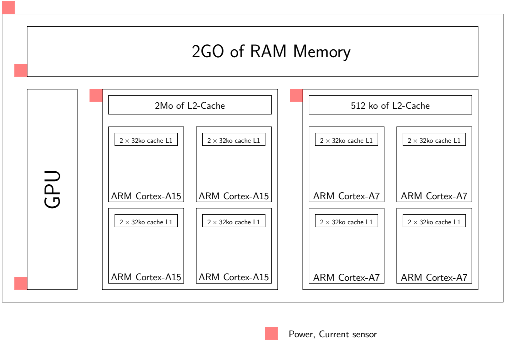
This presentation:

- How frequency, time and energy interfere between each other.
- From analysis to implementation
- Code generation for parallel real-time tasks

Plan

Time, power and energy on Arm big.LITTLE

- **Goal:** Build time and energy models
  - Executing different tasks on Xynos 5422.
  - With real-time priorities and under Linux

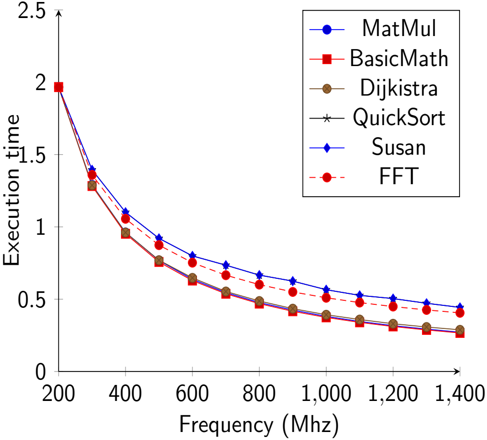


Experimentation Settings

- Modified from mi-bench: *Matrix Multiplication, Basic Math Operations, Dijkstra, Quick Sort, Susan-C, Fourier Transformations*.
- Using real-time priorities, SCHED_FIFO, each task is run for 1000 times.
- Modifying data size, task affinity, core frequency and core state.
- Measure the execution time, each core group power, total power.
- The modified benchmark is available on: `https://houssam.univ-lille1.fr/mi_bench_m.tar.gz`

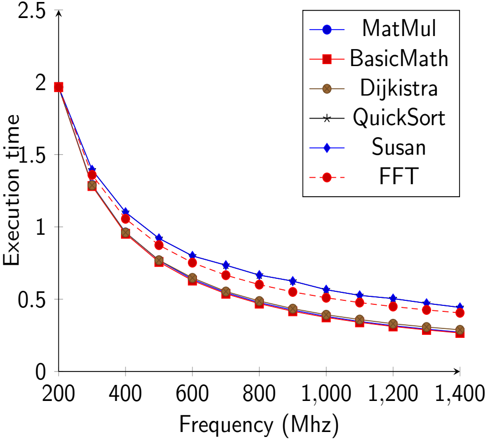
Time model

- One thread on one little core



Time model

- One thread on one little core

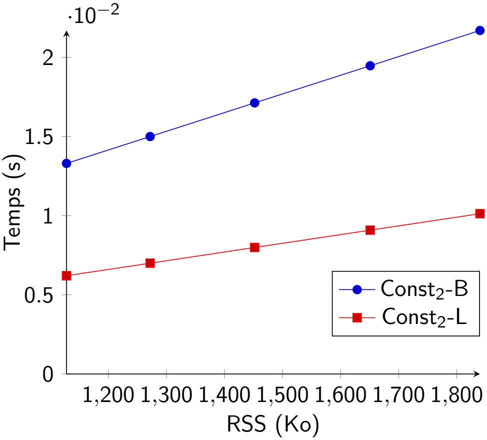


- Execution time is a function of the task code and the frequency
- Non linear regression: $\mathcal{C}(\mathrm{f}_{op}) = \frac{\mathrm{Const}_1}{\mathrm{f}_{op}} + \mathrm{Const}_2$

Execution time model

- Changing the processed data size

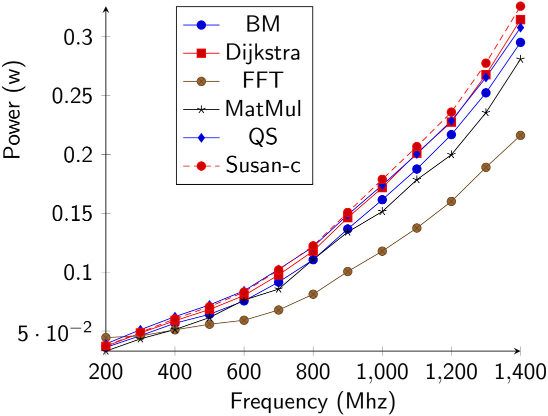
$$\mathcal{C}(\mathrm{f}_{op}) = \frac{\mathrm{Const}_1}{\mathrm{f}_{op}} + \mathrm{Const}_2$$



- *$Const_2$* represent the memory access under test conditions

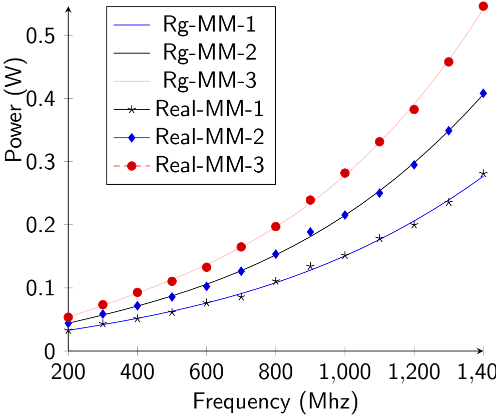
Power model

- One thread per core



- Power dissipation depends on the task

Power model: Regression and real values

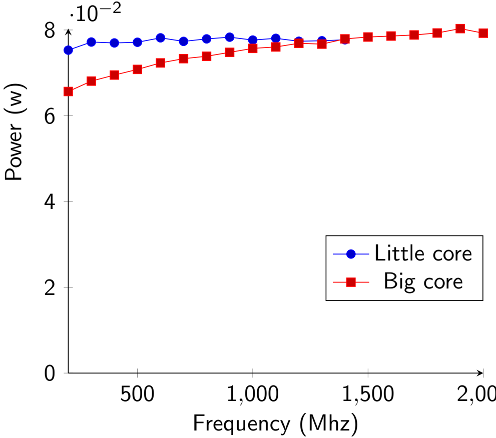


- 3^{rd} degree regression is applied.
- Regression is exact

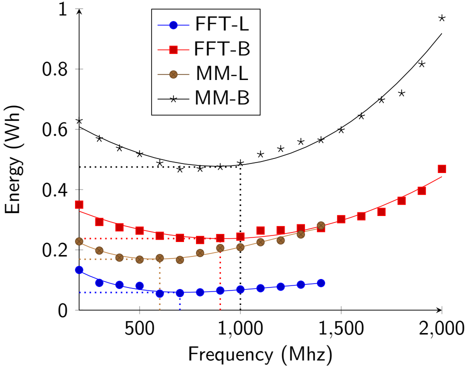
- Power dissipated by two (2) threads is not equal to twice ($\times 2$) the power dissipation of one thread.

Power dissipation: the memory

- The power dissipated by the memory of same processing: Little >> Big cores.
  - Smaller Cache-L2 ⇒ More cache-miss ⇒ More memory.access
- Memory power can reach more than 20% of a little core power (constant).



Energy model



- Raising the frequency "can" help to reduce the energy consumption until *effective* frequency.
- Effective frequency is task dependent.

What to remember

Power dissipation depends on:

- The core type (micro architecture) where the thread is allocated
- The operating frequency of the core
- The task itself

Hints:

- Two threads of the same task dissipate the same power (not energy)
- Static energy depends on the voltage which may depend on the frequency.
- Memory energy consumption is important.
- Effective frequency depends on a the task itself.

Plan

How analysis work: Introduction

```c
#include <stdlib.h>
#include <stdio.h>
void main(){
  main_th1:{
    a: {
      printf("Some initialization 1 \n");
      rt_1:
        printf("Real time processing 1 \n");
    }
    b:{
      printf("Some initizalition 2\n");
      rt_2:
        printf("Real time processing 2 \n");
    }
  }
}
```

begin processing: main_th1
T = 200
D = 100
label = init
begin processing: a
C = 21
label=init
begin processing: rt_1
label : rt
end processing: rt_1
end processing: a
begin processing: b
C = 21
label=init
begin processing: rt_2
label : rt
end processing: rt_2
end processing: b
end processing: main_th1

How it works: Input & Output

Inputs

- Real time inputs: Deadline, Period, execution time for each thread, etc.
- Parallelization inputs: The finest parallelization granularity.
- Energy coefficients for one parallel section

Outputs

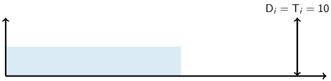
- Define the allocation of each thread to each core (source code)
- Select the operating frequency of core groups and core states

Analysis: task model

Analysis: task model

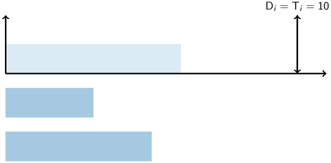
$$\mathcal{C}(f_{op}) = \frac{\text{Const}_1}{f_{op}} + \text{Const}_2$$

Analysis: task model



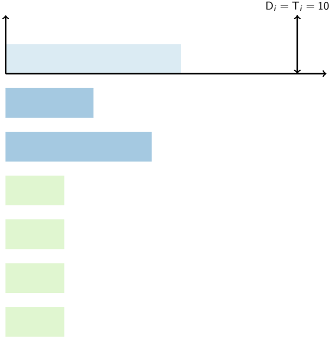
$$\mathcal{C}(f_{op}) = \frac{\mathrm{Const}_1}{f_{op}} + \mathrm{Const}_2$$

Analysis: task model



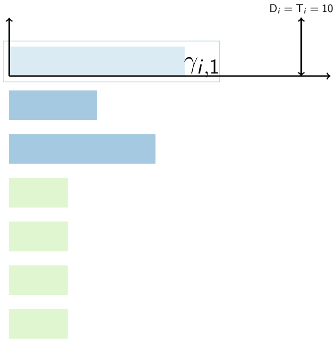
$$\mathcal{C}(f_{op}) = \frac{\mathrm{Const}_1}{f_{op}} + \mathrm{Const}_2$$

Analysis: task model



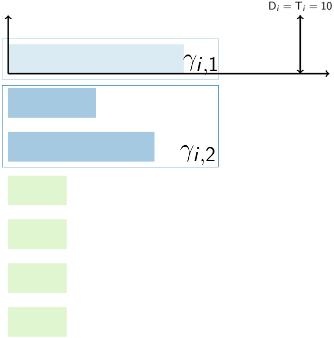
$$\mathcal{C}(f_{op}) = \frac{\text{Const}_1}{f_{op}} + \text{Const}_2$$

Analysis: task model



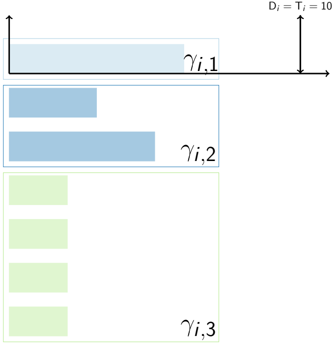
$$\mathcal{C}(\mathsf{f}_{op}) = \frac{\mathsf{Const}_1}{\mathsf{f}_{op}} + \mathsf{Const}_2$$

Analysis: task model



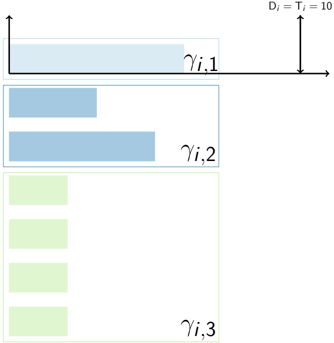
$$\mathcal{C}(f_{op}) = \frac{Const_1}{f_{op}} + Const_2$$

Analysis: task model



$$\mathcal{C}(f_{op}) = \frac{\text{Const}_1}{f_{op}} + \text{Const}_2$$

Analysis: task model

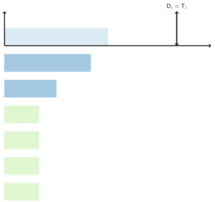


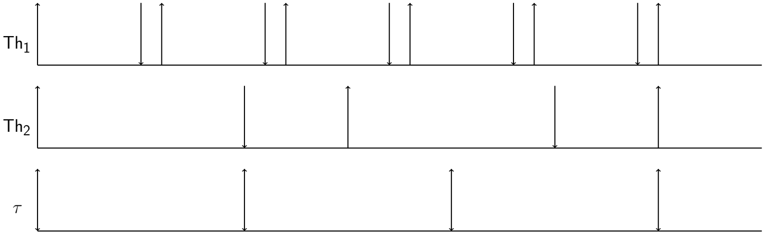
$$\mathcal{C}(f_{op}) = \frac{Const_1}{f_{op}} + Const_2$$

- Execution time of each thread
- $\sum_z C_{i,k,z} \geq C_{i,1,1}, \forall i, k$
- $\vec{\epsilon}$ an array of power dissipation coefficients.

Analysis: partitioning (1/2)

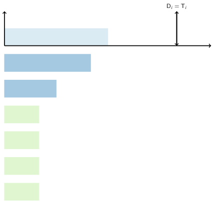
- Define the maximum execution time that can be allocated on the current core
- **Example**: Th_1, Th_2 are two threads allocated on the same core. We try to allocate τ on the same core
- Merge the threads that are meant to run onto the same cores

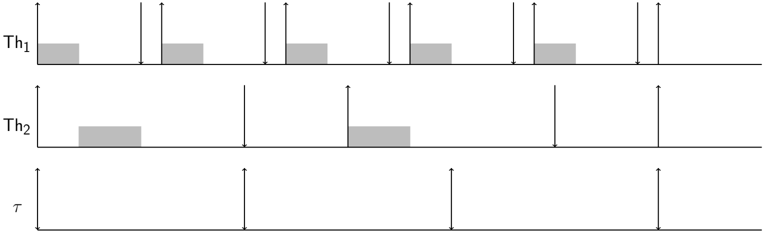




Analysis: partitioning (1/2)

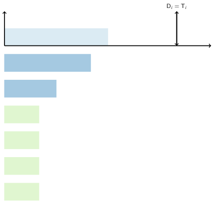
- Define the maximum execution time that can be allocated on the current core
- **Example**: Th_1, Th_2 are two threads allocated on the same core. We try to allocate τ on the same core
- Merge the threads that are meant to run onto the same cores

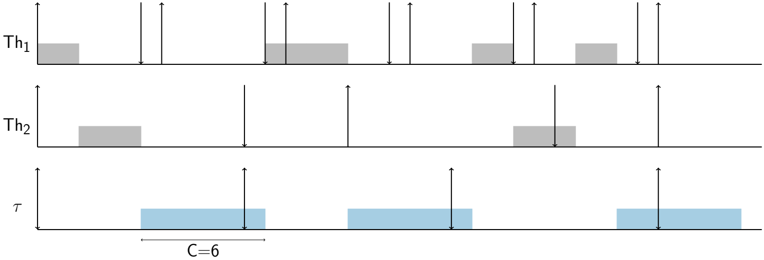




Analysis: partitioning (1/2)

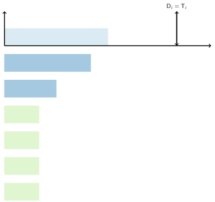
- Define the maximum execution time that can be allocated on the current core
- **Example**: Th_1, Th_2 are two threads allocated on the same core. We try to allocate τ on the same core
- Merge the threads that are meant to run onto the same cores

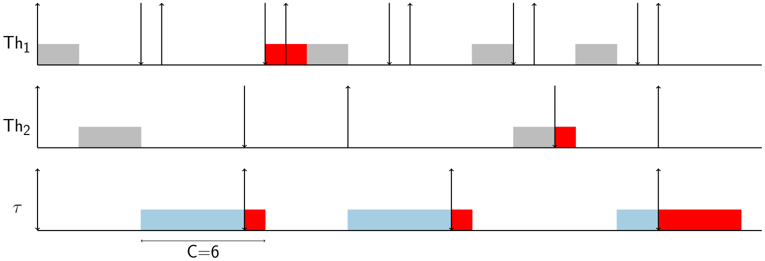




Analysis: partitioning (1/2)

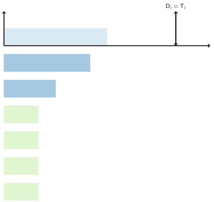
- Define the maximum execution time that can be allocated on the current core
- **Example**: $\mathrm{Th}_1, \mathrm{Th}_2$ are two threads allocated on the same core. We try to allocate τ on the same core
- Merge the threads that are meant to run onto the same cores

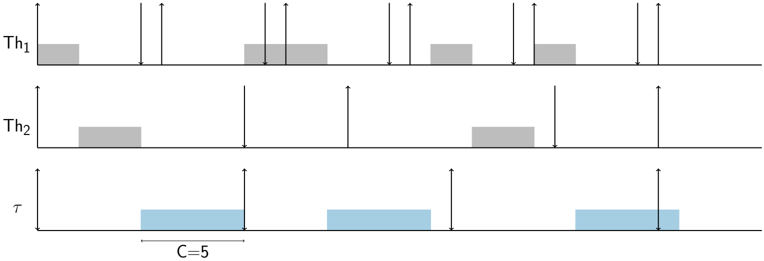




Analysis: partitioning (1/2)

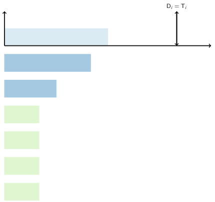
- Define the maximum execution time that can be allocated on the current core
- **Example**: $\mathrm{Th}_1, \mathrm{Th}_2$ are two threads allocated on the same core. We try to allocate τ on the same core
- Merge the threads that are meant to run onto the same cores

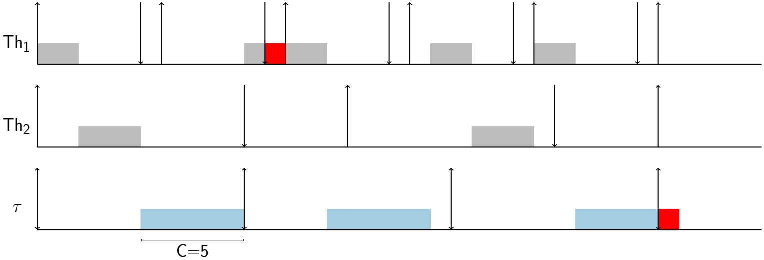




Analysis: partitioning (1/2)

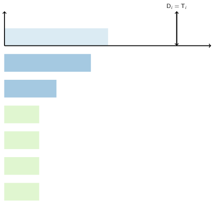
- Define the maximum execution time that can be allocated on the current core
- **Example**: Th_1, Th_2 are two threads allocated on the same core. We try to allocate τ on the same core
- Merge the threads that are meant to run onto the same cores

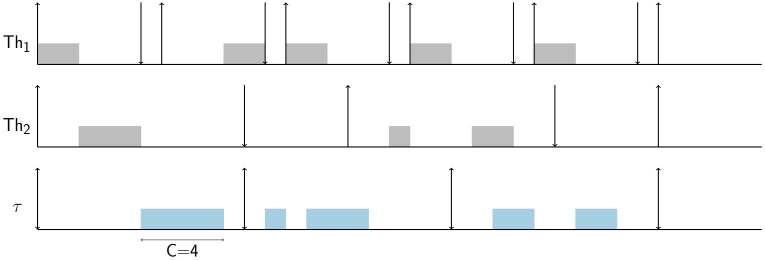




Analysis: partitioning (1/2)

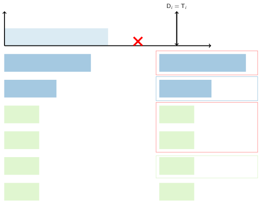
- Define the maximum execution time that can be allocated on the current core
- **Example**: Th_1, Th_2 are two threads allocated on the same core. We try to allocate τ on the same core
- Merge the threads that are meant to run onto the same cores

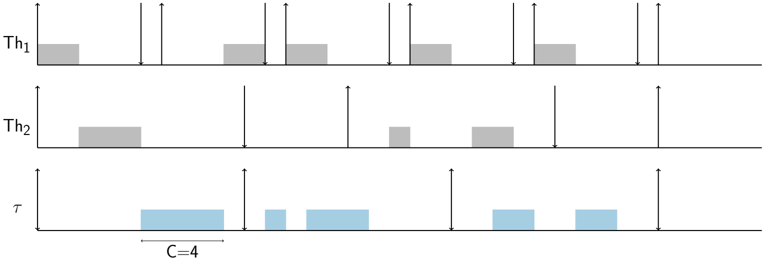




Analysis: partitioning (1/2)

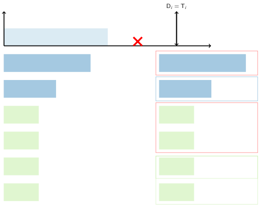
- Define the maximum execution time that can be allocated on the current core
- **Example**: Th_1, Th_2 are two threads allocated on the same core. We try to allocate τ on the same core
- Merge the threads that are meant to run onto the same cores

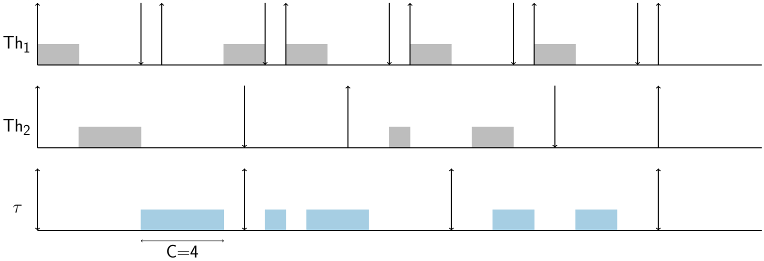




Analysis: partitioning (1/2)

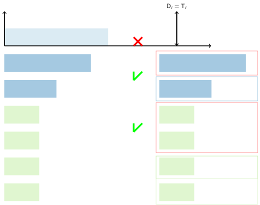
- Define the maximum execution time that can be allocated on the current core
- **Example**: $\mathrm{Th}_1, \mathrm{Th}_2$ are two threads allocated on the same core. We try to allocate τ on the same core
- Merge the threads that are meant to run onto the same cores



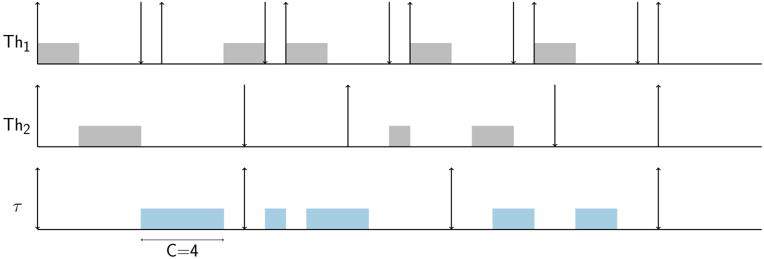


Analysis: partitioning (1/2)

- Define the maximum execution time that can be allocated on the current core
- **Example**: $\mathrm{Th}_1, \mathrm{Th}_2$ are two threads allocated on the same core. We try to allocate τ on the same core
- Merge the threads that are meant to run onto the same cores

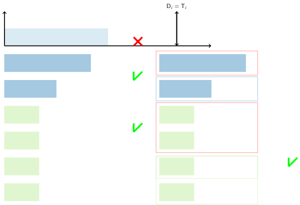


Threads allocated on current core

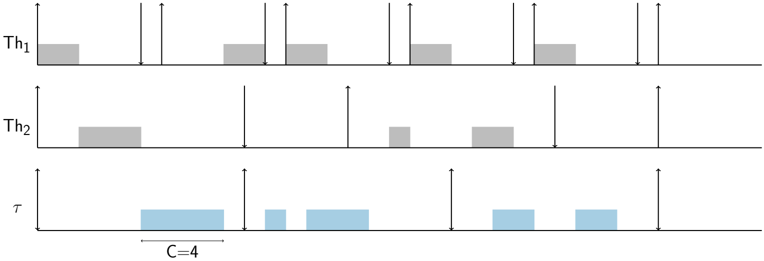


Analysis: partitioning (1/2)

- Define the maximum execution time that can be allocated on the current core
- **Example**: Th_1, Th_2 are two threads allocated on the same core. We try to allocate τ on the same core
- Merge the threads that are meant to run onto the same cores



Threads allocated on current core



Analysis: partitioning (2/2)

1. Select a task and core
2. Test the feasibility of all threads on the same core
3. If feasible allocate all threads on the same core
4. Else: Compute excess
   4.1 if excess is between 0 and the task sequential execution
       - split and allocate only a sub-task
   4.2 Else: select an other core
5. if all core are investigated, abort scheduling

References

- Zahaf, Lipari "Energy-efficient scheduling for moldable real-time tasks on heterogeneous computing platforms" , Journal of Systems Architecture, 2017
- H.E. Zahaf, A.E Benyamina , R. Olejnik, G. Lipari, Pierre Boulet "Modeling parallel task with Di-Graphs" , RTNS'2016, Brest France

Example (1/3)

```c
#include <stdlib.h>
#include <stdio.h>
void main(){
  main_th1:{
    a: {
      printf("Some initialization 1 \n");
      rt_1:
        printf("Real time processing 1 \n");
    }
    b:{
      printf("Some initizalition 2\n");
      rt_2:
        printf("Real time processing 2 \n");
    }
  }
}
```

begin processing: main_th1
T = 200
D = 100
label = init
begin processing: a
C = 21
label=init
begin processing: rt_1
label : rt
end processing: rt_1
end processing: a
begin processing: b
C = 21
label=init
begin processing: rt_2
label : rt
end processing: rt_2
end processing: b
end processing: main_th1

Example (2/3): 2 threads on 2 cores

```
void * main_th1_a(void *arg){
  // real-time setting times
  struct periodique *cp = (struct periodique *) arg;
  struct timespec Begin,END;
  struct timespec T = ms_tospec(cp->periode);
  struct timespec D = ms_tospec(cp->deadline);
  int cond=1;
  int MissedDeadlines = 0 ;
  // affinity setting
  int cpu = 0;
  cpu_set_t cpuset;
  CPU_ZERO(&cpuset);
  CPU_SET( cpu , &cpuset);
  sched_setaffinity(0, sizeof(cpuset), &cpuset);
  // user init
  printf("Some initialization 1 \n");
  while(cond){
    // included header
    clock_gettime(CLOCK_REALTIME, &Begin);
    struct timespec NA = timespec_add(&Begin,&T);
    struct timespec Dij= timespec_add(&Begin,&D);
    // The real-time processing
    printf("Real time processing 1 \n");
    // included footer
    clock_gettime(CLOCK_REALTIME, &End);
    clock_nanosleep(CLOCK_REALTIME,
                    TIMER_ABSTIME, &NA, NULL);
  }
}
```

```
void * main_th1_a(void *arg){
  // real-time setting times
  struct periodique *cp = (struct periodique *) arg;
  struct timespec Begin,End;
  struct timespec T = ms_tospec(cp->periode);
  struct timespec D = ms_tospec(cp->deadline);
  int cond=1;
  int MissedDeadlines = 0 ;
  // affinity setting
  int cpu = 1;
  cpu_set_t cpuset;
  CPU_ZERO(&cpuset);
  CPU_SET( cpu , &cpuset);
  sched_setaffinity(0, sizeof(cpuset), &cpuset);
  // user init
  printf("Some initialization 2 \n");
  while(cond){
    // included header
    clock_gettime(CLOCK_REALTIME, &Begin);
    struct timespec NA = timespec_add(&Begin,&T);
    struct timespec Dij= timespec_add(&Begin,&D);
    // The real-time processing
    printf("Real time processing 2 \n");
    // included footer
    clock_gettime(CLOCK_REALTIME, &End);
    clock_nanosleep(CLOCK_REALTIME,
                    TIMER_ABSTIME, &NA, NULL);
  }
}
```

Example (2/3): 2 threads on 1 core

```
void * main_th1_a(void *arg){
  // real-time setting times
          ...
  // user init
  printf("Some initialization 1 \n");
  printf("Some initialization 2 \n");
  while(cond){
    // included header
          ...
    // The real-time processing
    printf("Real time processing 1 \n");
    printf("Real time processing 2 \n");
    // included footer
             ...
  }
}
```

Frequency selection

- Operating frequency is selected between minimal effective frequency and max. frequency.
- Computing strength is defined as: $\frac{\mathrm{f}_{\mathrm{op}}^{g}}{\mathrm{f}_{max}}\mathrm{nb}$

Frequency selection

- Operating frequency is selected between minimal effective frequency and max. frequency.
- Computing strength is defined as: $\frac{\mathrm{f}_{\mathrm{op}}^{g}}{\mathrm{f}_{max}}\mathrm{nb}$

Frequency selection

- Operating frequency is selected between minimal effective frequency and max. frequency.
- Computing strength is defined as: $\frac{\mathrm{f}_{\mathrm{op}}^{g}}{\mathrm{f}_{max}}\mathrm{nb}$

Frequency selection

- Operating frequency is selected between minimal effective frequency and max. frequency.
- Computing strength is defined as: $\frac{f_{op}^{g}}{f_{max}} nb$

Frequency selection

- Operating frequency is selected between minimal effective frequency and max. frequency.
- Computing strength is defined as: $\frac{f^{g}_{op}}{f_{max}}nb$

Frequency selection

- Operating frequency is selected between minimal effective frequency and max. frequency.
- Computing strength is defined as: $\frac{f^g_{op}}{f_{max}} nb$

Frequency selection

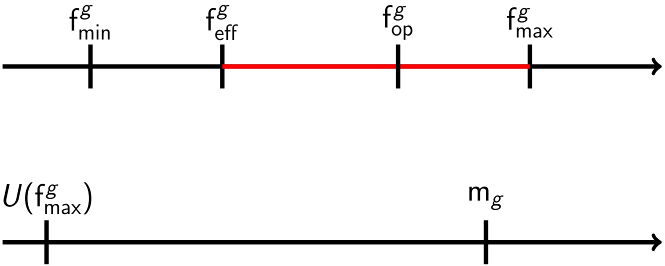
- Operating frequency is selected between minimal effective frequency and max. frequency.
- Computing strength is defined as: $\frac{f_{op}^{g}}{f_{max}}nb$

Frequency selection

- Operating frequency is selected between minimal effective frequency and max. frequency.
- Computing strength is defined as: $\frac{f_{op}^{g}}{f_{max}} nb$

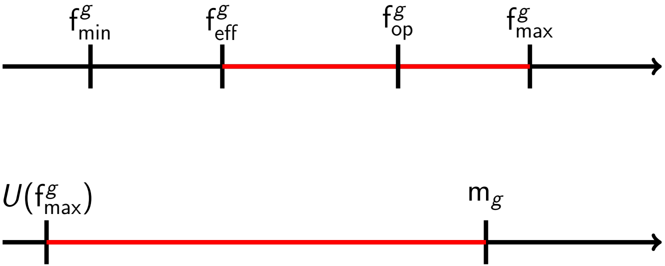
Frequency selection

- Operating frequency is selected between minimal effective frequency and max. frequency.
- Computing strength is defined as: $\frac{\mathrm{f}_{\mathrm{op}}^{g}}{\mathrm{f}_{max}}\mathrm{nb}$



Frequency selection

- Operating frequency is selected between minimal effective frequency and max. frequency.
- Computing strength is defined as: $\frac{f^g_{op}}{f_{max}} nb$

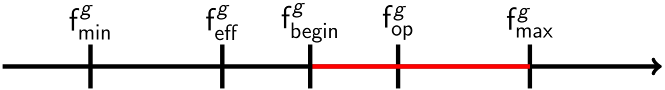


Frequency selection

- Operating frequency is selected between minimal effective frequency and max. frequency.
- Computing strength is defined as: $\frac{f^g_{op}}{f_{max}}nb$

Frequency selection

- Operating frequency is selected between minimal effective frequency and max. frequency.
- Computing strength is defined as: $\frac{\mathrm{f}_{\mathrm{op}}^{g}}{\mathrm{f}_{max}}\mathrm{nb}$



Platform

The platform allows :

- Synchronize threads on different cores wake up (more accuracy)
- Using SCHED_DEADLINE with budget C/T_i
- User may select options in code generation
- Allow variable sharing (but not taken into account in analysis)
- Allow to implicitly parallelize iteration-independent for loops

Plan

Conclusions and future work

Conclusions

- Parallelization can help to achieve better feasibility and energy efficiency
- Better to go sequential when feasible
- Selected frequency very low $\rightarrow$ pack tasks and *turn* off cores

Open questions

- How to adapt frequency selection when running tasks with different profiles?