Scheduling on Low-Power Multi- and Many-Cores

Ben Juurlink

Computer Engineering Laboratory, Delft University of Technology

4-6-2009

Agenda

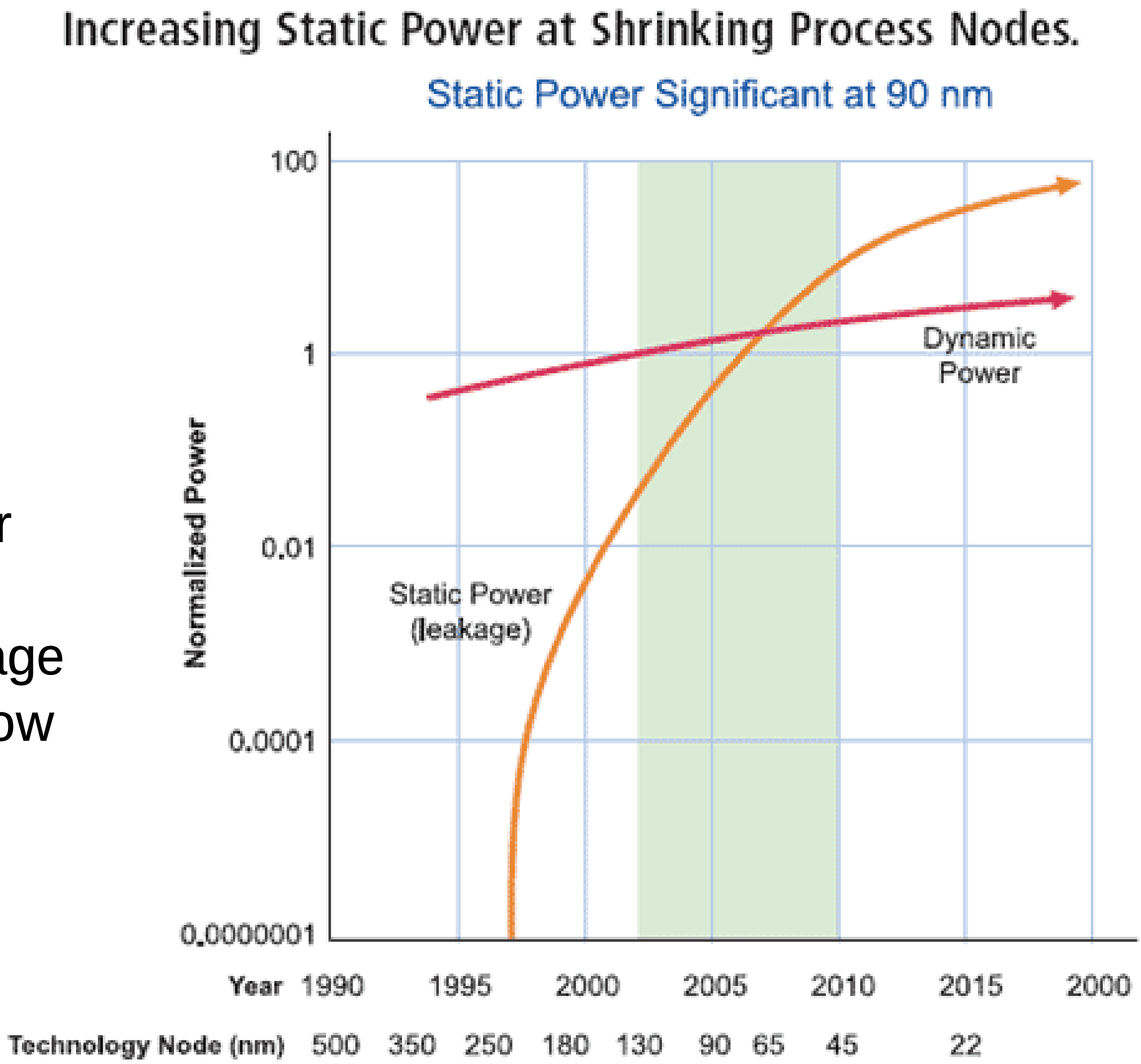
- Part I: Leakage-Aware Multiprocessor Scheduling
- Part II: Scheduling issues in a highly scalable parallel implementation of H.264 decoding

Part I: Leakage-Aware Multiprocessor Scheduling

- Motivation
- Power/Energy Consumption
- Dynamic Voltage/Frequency Scaling (DVFS)
- Processor Shutdown
- System and application model
- Schedule & Stretch (S&S)
- Leakage-Aware Multiprocessor Scheduling
- LIMIT
- Experimental Results
- Conclusions

Motivation

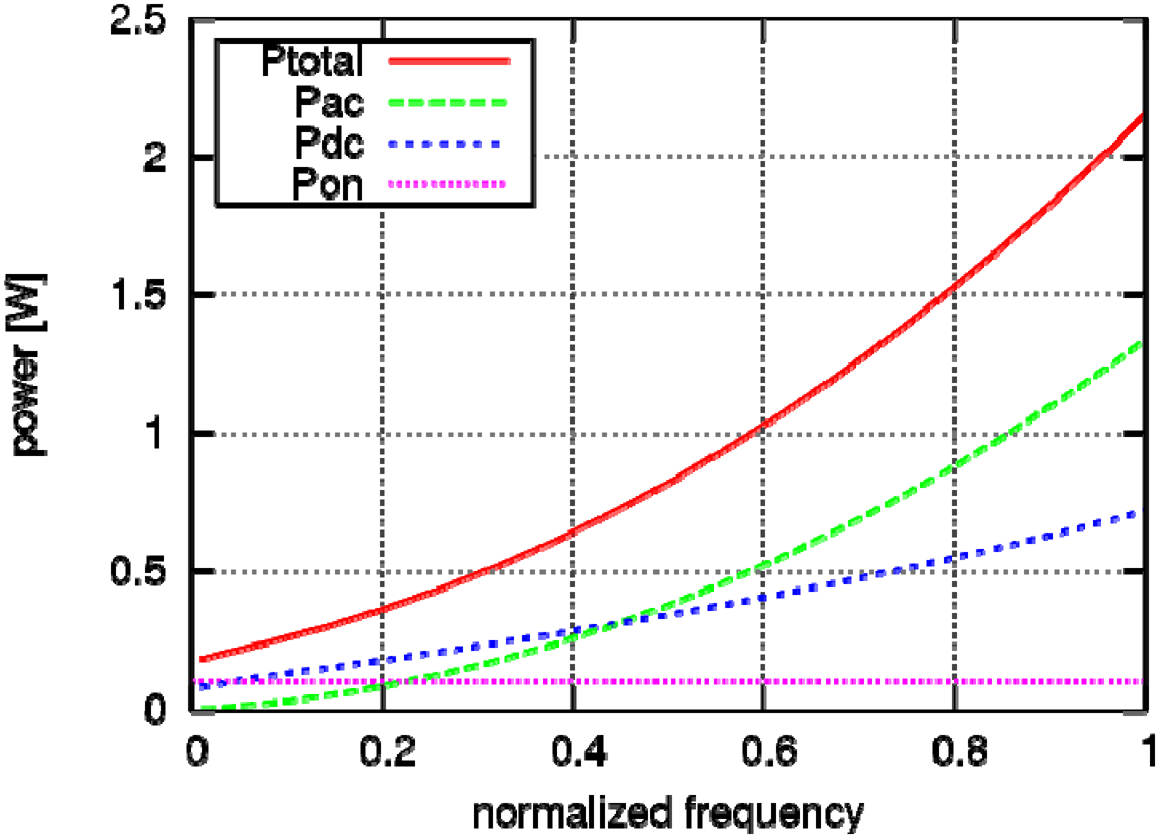
- Currently, dynamic power dominates static power
- Static power due to leakage current is expected to grow significantly



Source: http://www.actel.com

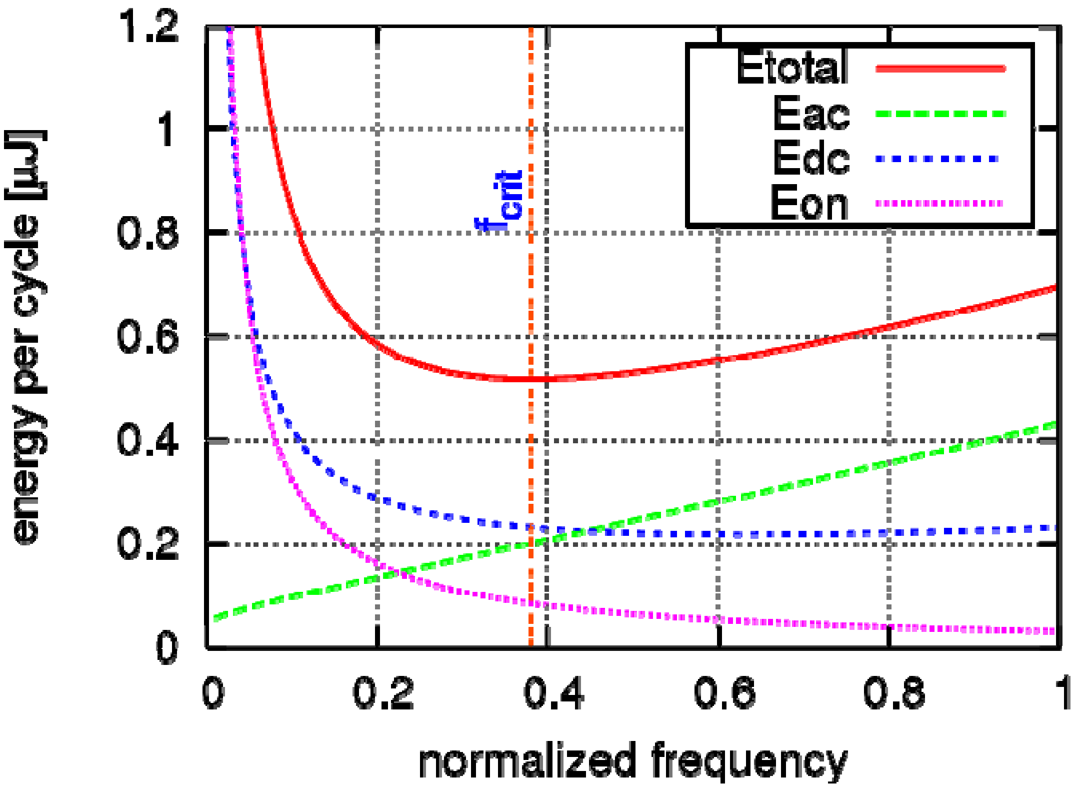
Power Consumption

Power model of (Jejurikar et al., 2004), 70nm technology



Energy Consumption

- Scaling below critical frequency f_{crit} (normalized 0.38, actual 1.18GHz) increases energy consumption



Dynamic Voltage/Frequency Scaling (DVFS)

$$P = \underbrace{\alpha \bullet C_{eff} \bullet V_{dd}^2 \bullet f}_{\text{dynamic power } P_{\mathrm{AC}}} + \underbrace{V_{dd} \bullet I_{subn} + |V_{bs}| \bullet I_j}_{\text{static power } P_{\mathrm{DC}}} + P_{on}$$

- Dynamic power grows quadratically with supply voltage
- Static power grows “linearly” with supply voltage
- $V = \beta_1 + \beta_2 \bullet f$
- Static energy consumption increases when voltage is scaled down

Processor Shutdown

$$P = \underbrace{\alpha \bullet C_{eff} \bullet V_{dd}^2 \bullet f}_{\text{dynamic power}} + \underbrace{V_{dd} \bullet I_{subn} + |V_{bs}| \bullet I_j + P_{on}}_{\text{static power}}$$

- Processor shutdown reduces both static and dynamic energy consumption

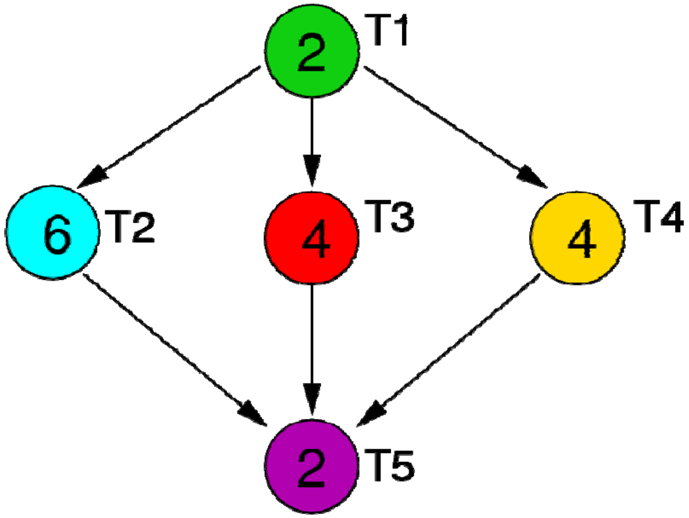
- Shutdown involves an (energy) penalty due to loss of state (caches, branch predictors)
  - ≈483 μJ (Jejurikar et al., 2004)
  - Shutdown saves energy only if idle period sufficiently long

System Model

- Shared memory multi-core
- Application computation bound
- Scaling down clock frequency by factor of *k* increases execution time by factor of at most *k*

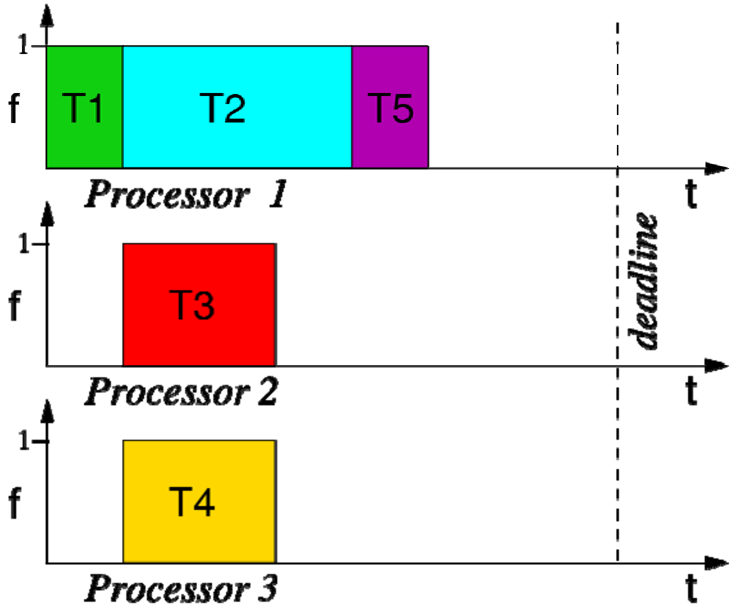
Application Model

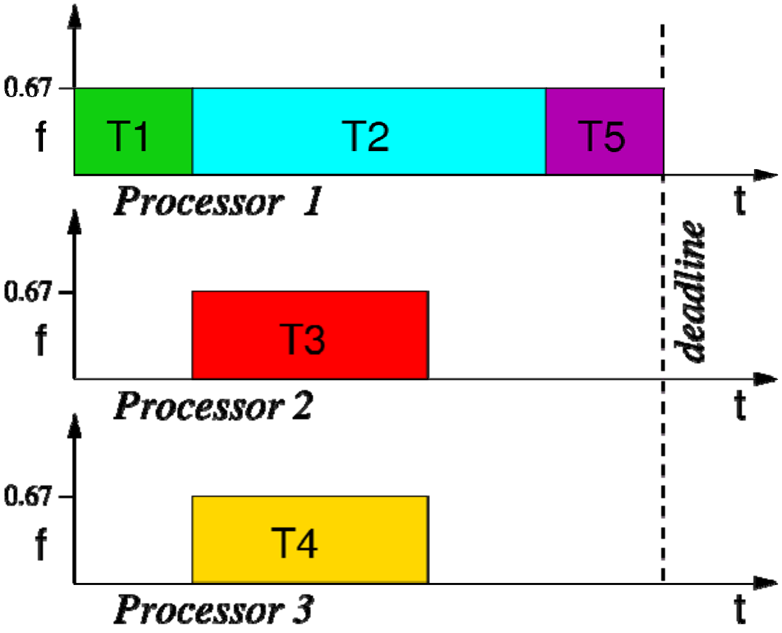
- Weighted directed acyclic graph $G = (V, E, w)$
- Graphs taken from Standard Task Graph Set (http://www.kasahara.elec.waseda.ac.jp/schedule/)
    - random TGs
    - application TGs
- Deadlines relative to critical path length (CPL)
    - Coarse-grain tasks: 1 unit = 1 ms at max frequency ($3.1 \cdot 10^6$ cycles)
    - Fine-grain tasks: 1 unit = 10 μs at max frequency ($3.1 \cdot 10^4$ cycles)



Schedule and Stretch

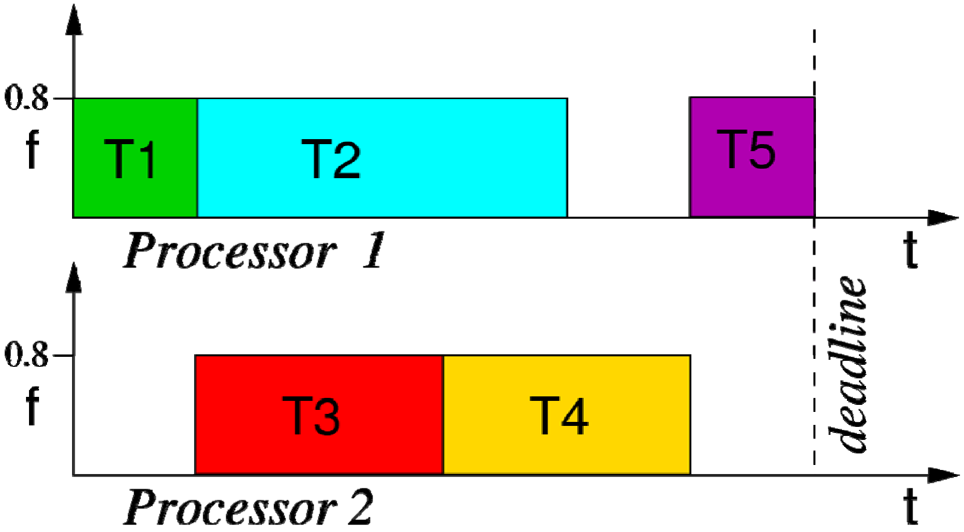
- When dynamic power dominates, optimal strategy is to
  - schedule tasks on as many processors as can be used to reduce makespan (we employ LS+EDF)
  - use remaining time at end of schedule (slack) to lower voltage/ frequency as much as possible
  - Due to (Zhu et al., 2003) and (Gruian and Kuchcinski, 2001)





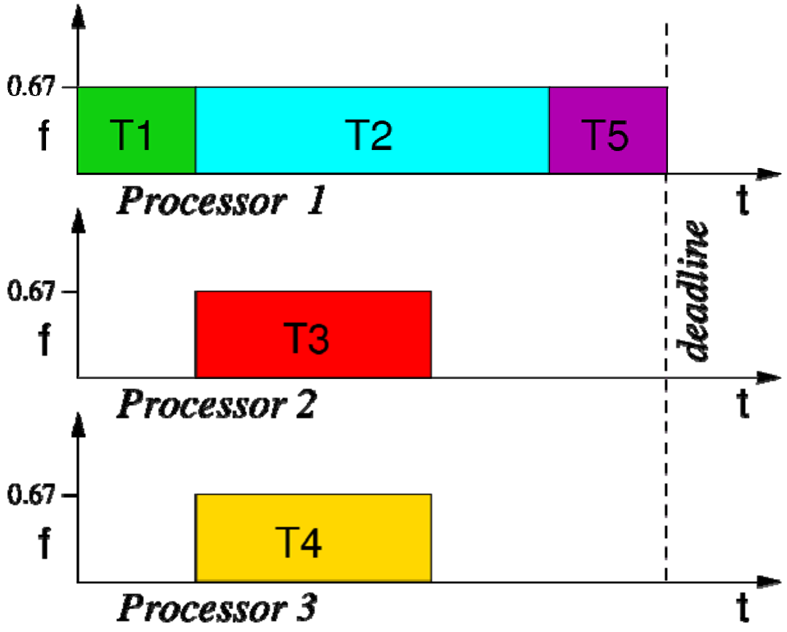
Leakage-Aware Multiprocessor Scheduling

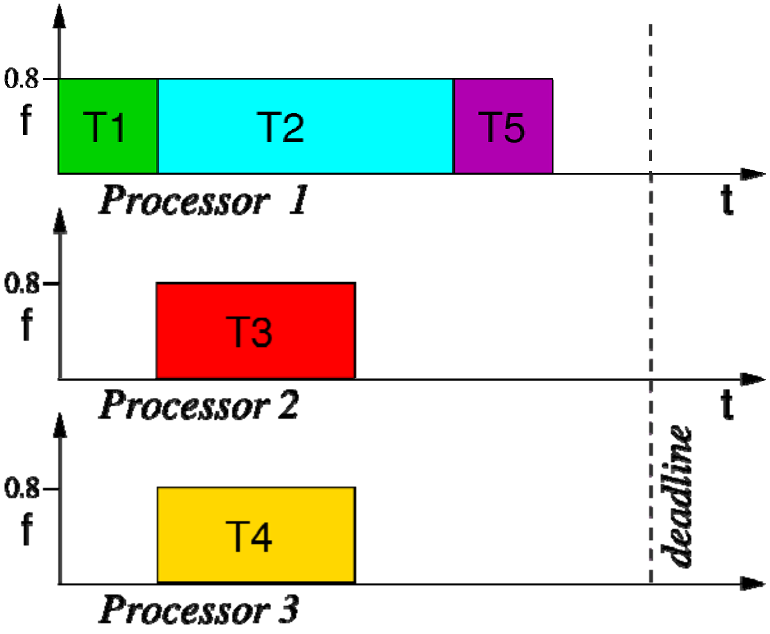
- When dynamic power does not dominate, need to find balance between
  - number of processors employed
  - amount of voltage/frequency scaling
- Our LAMPS (💡) algorithm:
  - for each number of processors N_{min} … N_{max}
    - schedule using EDF
    - use slack at end of schedule to lower voltage/frequency
  - return number of cores with least energy consumption



S&S+PS

- Schedule to minimize makespan
- Compute energy consumption for each voltage/frequency level
  - shutdown cores during idle periods if it reduces energy
- Return voltage/frequency level with least energy consumption





LAMPS+PS

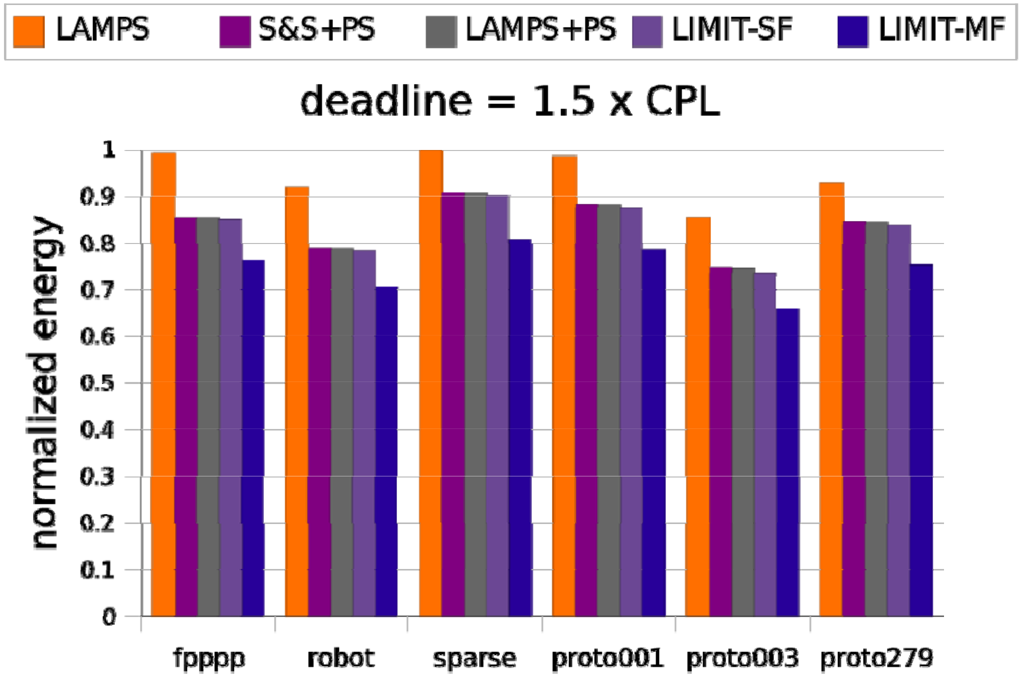
- For each number of processors N_{min} ... N_{max}
  - Schedule using LS+EDF
  - Compute energy consumption for each voltage/frequency level
    - shutdown cores during idle periods if it reduces energy
- Return voltage/frequency level with least energy consumption

- LAMPS+PS determines an optimal balance between
  - voltage/frequency scaling
  - processor shutdown
  - number of cores to employ

How close to optimal?

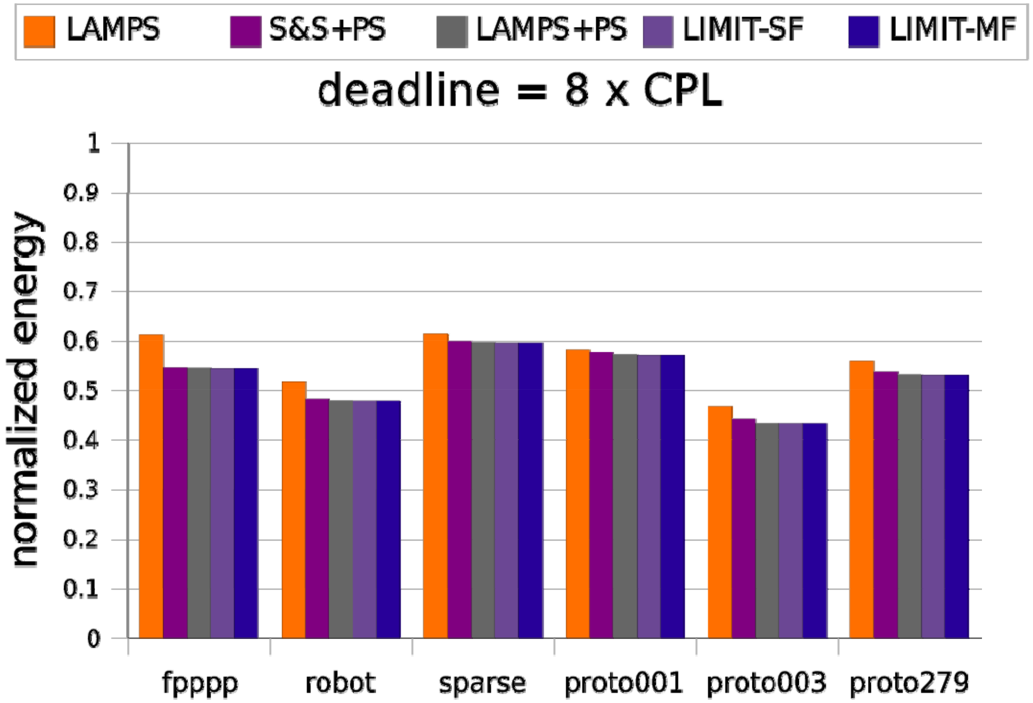
- Known limitations:
  - EDF is “just” a heuristic
  - In our low-energy scheduling algorithms, all processors run at same frequency and this frequency is constant throughout the schedule
- Lower bounds:
  - Idle cores consume no energy
  - Number of cores = number of tasks
  - LIMIT-SF: All cores are scaled down to critical frequency, or as much as possible to meet deadline → no single-frequency schedule can consume less energy
  - LIMIT-MF: All cores are scaled down to critical frequency, possibly missing deadline → no schedule can consume less energy

Experimental Results (I)



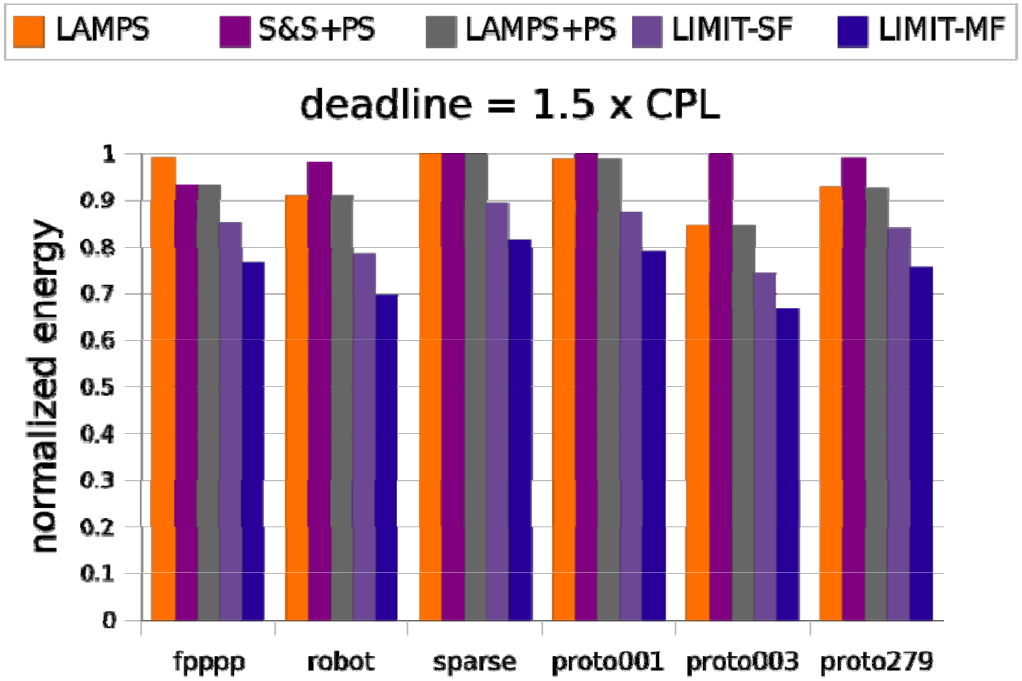
- For coarse-grain tasks and tight deadlines:
  - LAMPS performs just little better than S&S (cannot use fewer cores)
  - Processor shutdown approaches perform better (sufficient intra-schedule slack) and almost as good as LIMIT-SF
  - LIMIT-MF lower bound probably too tight in this case (misses deadlines)

Experimental Results (II)



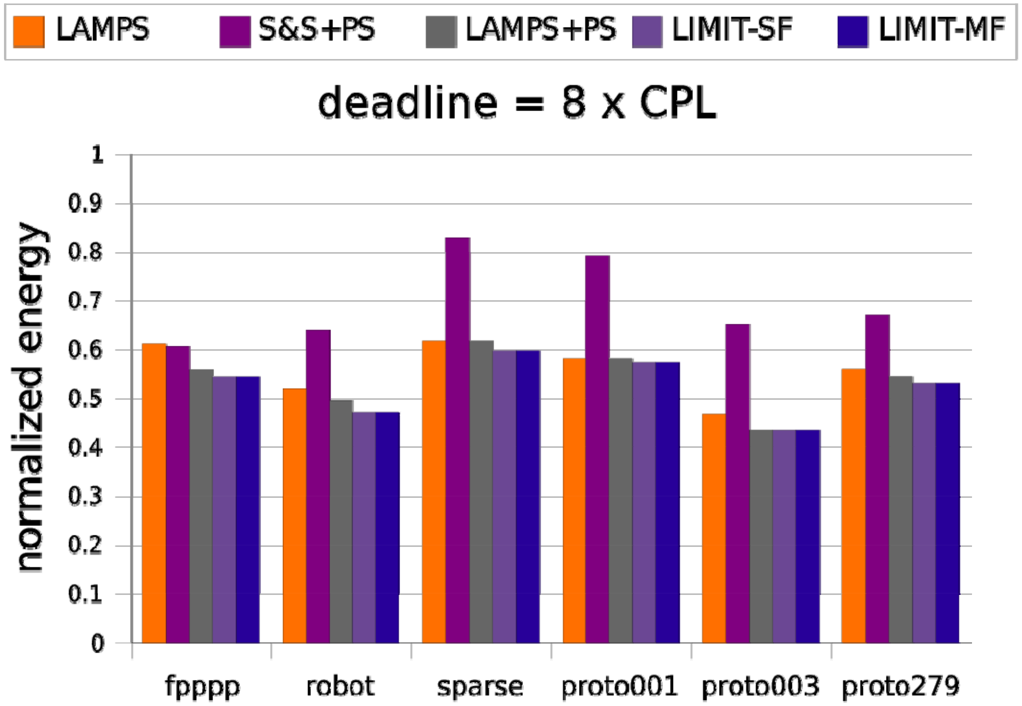
- For coarse-grain tasks and loose deadlines:
  - LAMPS much better than S&S (can employ fewer cores)
  - Processor shutdown approaches perform only slightly better than LAMPS (can use intra-schedule slack to shutdown cores or to reduce number of cores)
  - LAMPS+PS optimal

Experimental Results (III)



- For fine-grain tasks and tight deadlines:
  - LAMPS significantly better than S&S only in few cases (when not all cores are needed to meet deadline)
  - S&S+PS worse than LAMPS (insufficient intra-schedule slack)
  - Quite a gap between LAMPS+PS and LIMIT-SF/LIMIT-MF (room for improvement or lower bounds too tight)

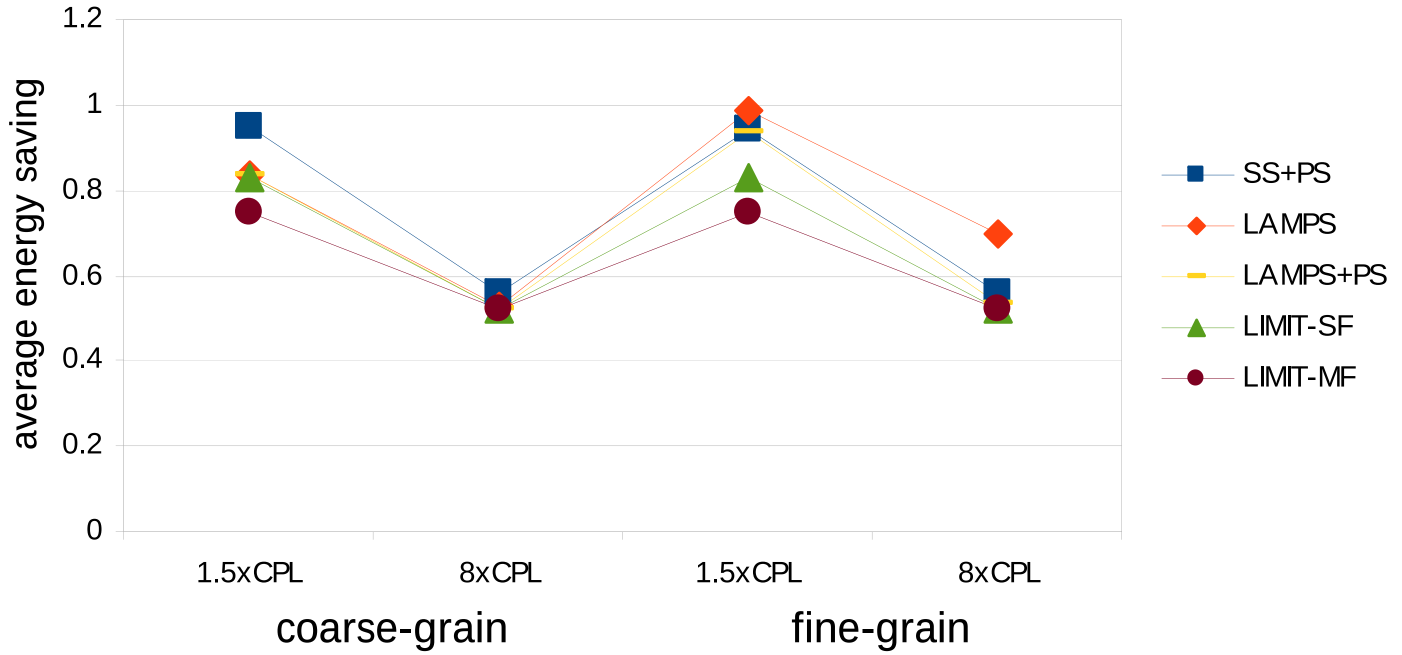
Experimental Results (IV)



- For fine-grain tasks and loose deadlines:
  - LAMPS much better than S&S and S&S+PS (insufficient intra-schedule slack)
  - LAMPS+PS close to optimal

Conclusions

- When leakage-current is significant, the possibility of reducing energy by only employing DVFS is limited
- In this case, higher energy savings are obtained by shutting down cores temporarily or completely
- For coarse-grain tasks, LAMPS+PS attains ≥ 84% of possible energy saving



Future Work

- Determine a stronger lower bound (can be formulated as ILP problem)
- If results show that higher energy savings can be obtained, develop a scheduling algorithm that maximizes amount of slack
- Incorporate communication
- Other scheduling models
- How to deal w/ incomplete information (worst-case vs. actual execution time)
- ...

Part II: Scheduling Issues in a Highly Scalable Parallel Implementation of H.264 Decoding

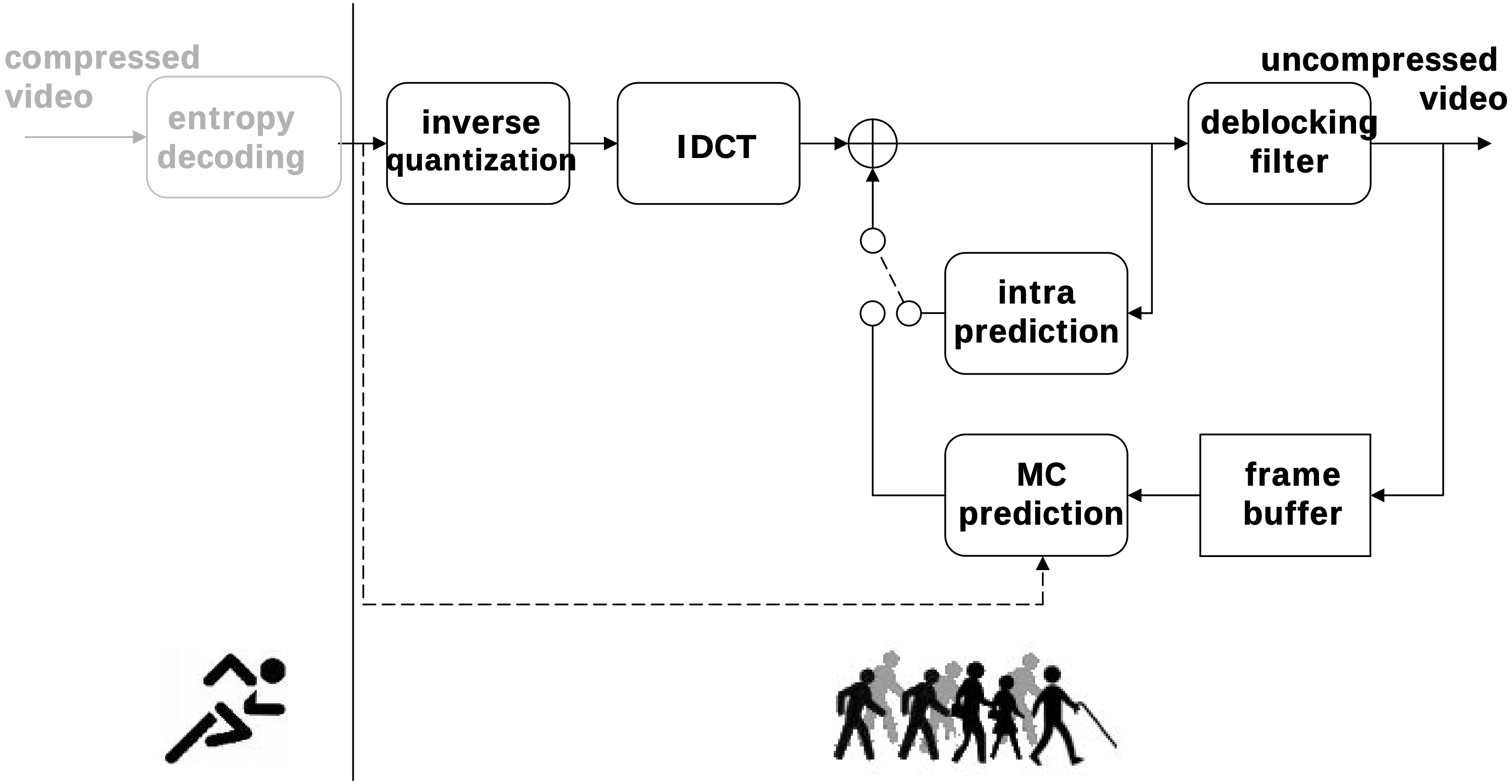
- Motivation
- H.264 decoding
  - where's the parallelism?
- 2D-Wave
  - need for dynamic scheduling
  - parallel programming model
  - 2D-Wave pseudo-code
  - user-level scheduling for locality
  - scalability
- 3D-Wave
  - implementation
  - scalability
- Increasing programmability
  - ENCORE project
- Conclusions

Motivation

- “*Developing parallel applications to harness and effectively use the massively parallel tera-scale processors is likely to be the key challenge for tera-scale computing*.” (Azimi et al., Intel Technology Journal, 2007)

- As a case study, we consider H.264 decoding
  - State-of-the-art video coding standard
  - Challenging to find massive TLP

Overview of H.264

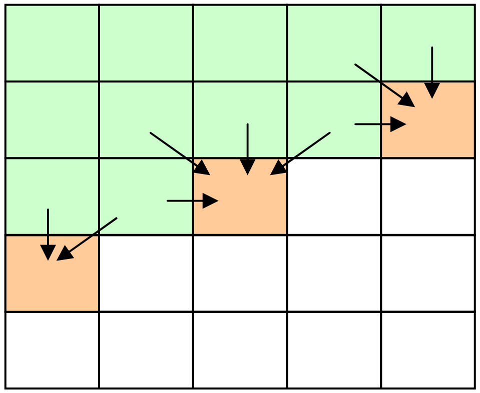


Where is the Data Parallelism?

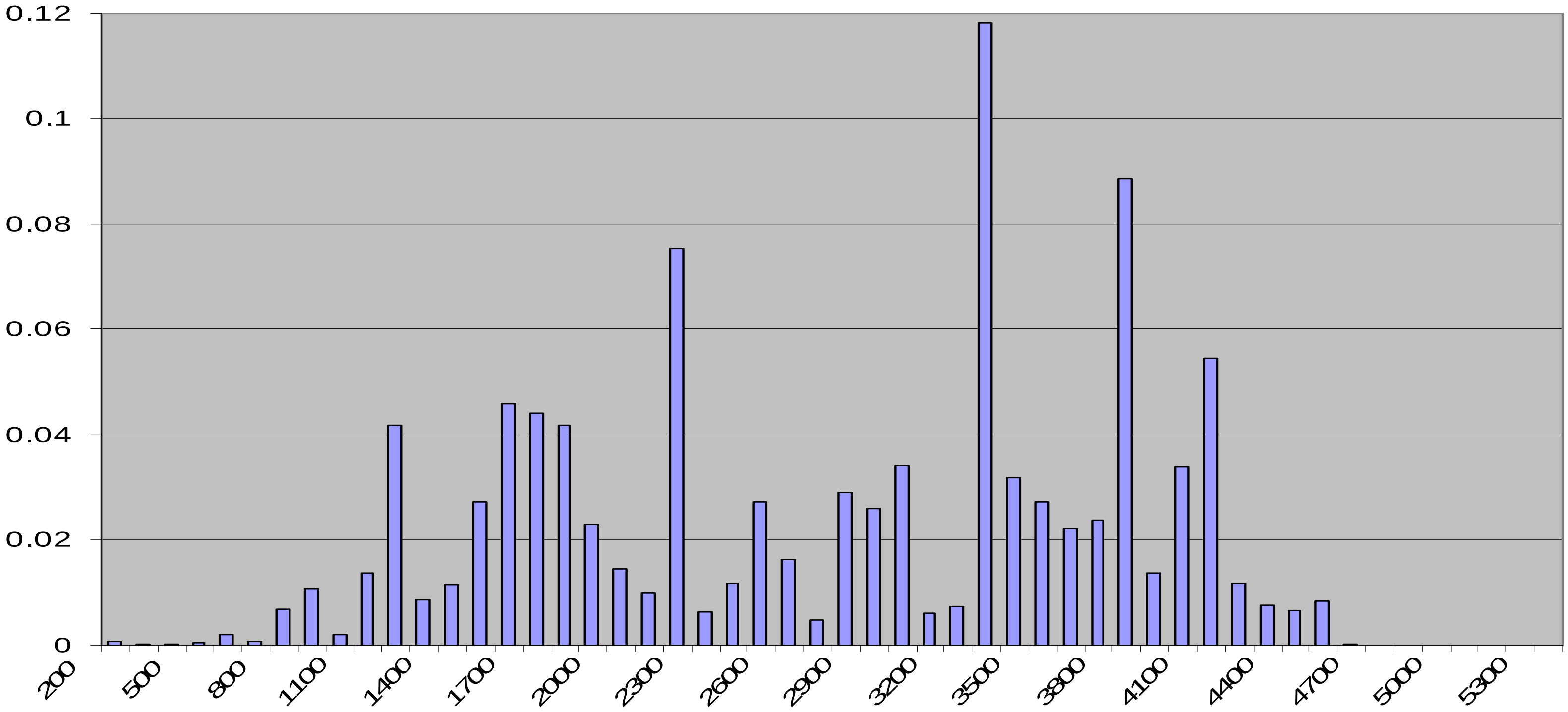
- Between frames?
  - Limited, because of inter-frame dependences
- Between slices?
  - No, because there might be only one slice per frame
- Between macroblocks (MBs)?
  - Yes
- Between operations?
  - Of course. ILP and SIMD (short vectors).

2D-Wave

- Proposed by (Van der Tol et al., 2003)
- Exploits intra-frame MB-level parallelism



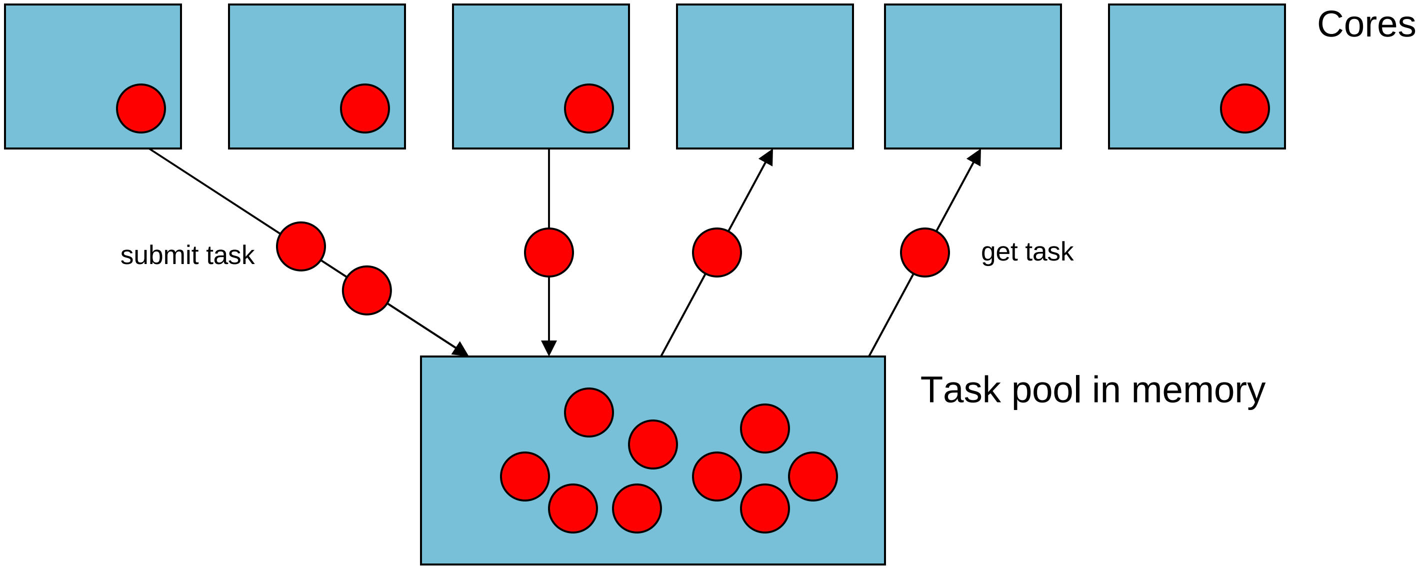
The Need for Dynamic Scheduling



Cycle Distribution for PicturePrediction() on NXP’s TriMedia

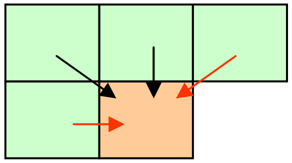
Parallel Programming Model: Task Pool

- Software structure in shared memory
- Contains tasks ready for execution



2D-Wave: Deblocking a Frame

- MB dependencies covered by dependencies from upper-right MB to current MB and from left MB to current MB



```
int deblock_ready[w][h];   // array of reference counts


void deblock_frame()
{
  for(x = 1; x <= w; x++)
    for(y = 1; y <= h; y++)
      deblock_ready[x][y] = initial reference count; // 0, 1, or 2

  tp_submit(deblock_mb, 1, 1); // start first task: MB <1,1>

  tp_wait();
}
```

2D-Wave: Deblocking a Macroblock

```
void deblock_mb(int x, int y)
{
   ... the actual work ...

   if(x >= 2 && y != h)
   {
     new_value = tp_atomic_decrement(&deblock_ready[x-1][y+1], 1);
     if(new_value == 0)
       tp_submit(deblock_mb, x - 1, y + 1);
   }

   if(x != w)
   {
     new_value = tp_atomic_decrement(&deblock_ready[x+1][y], 1);
     if(new_value == 0)
       tp_submit(deblock_mb, x + 1, y);
   }
}
```

User-level Scheduling for Locality

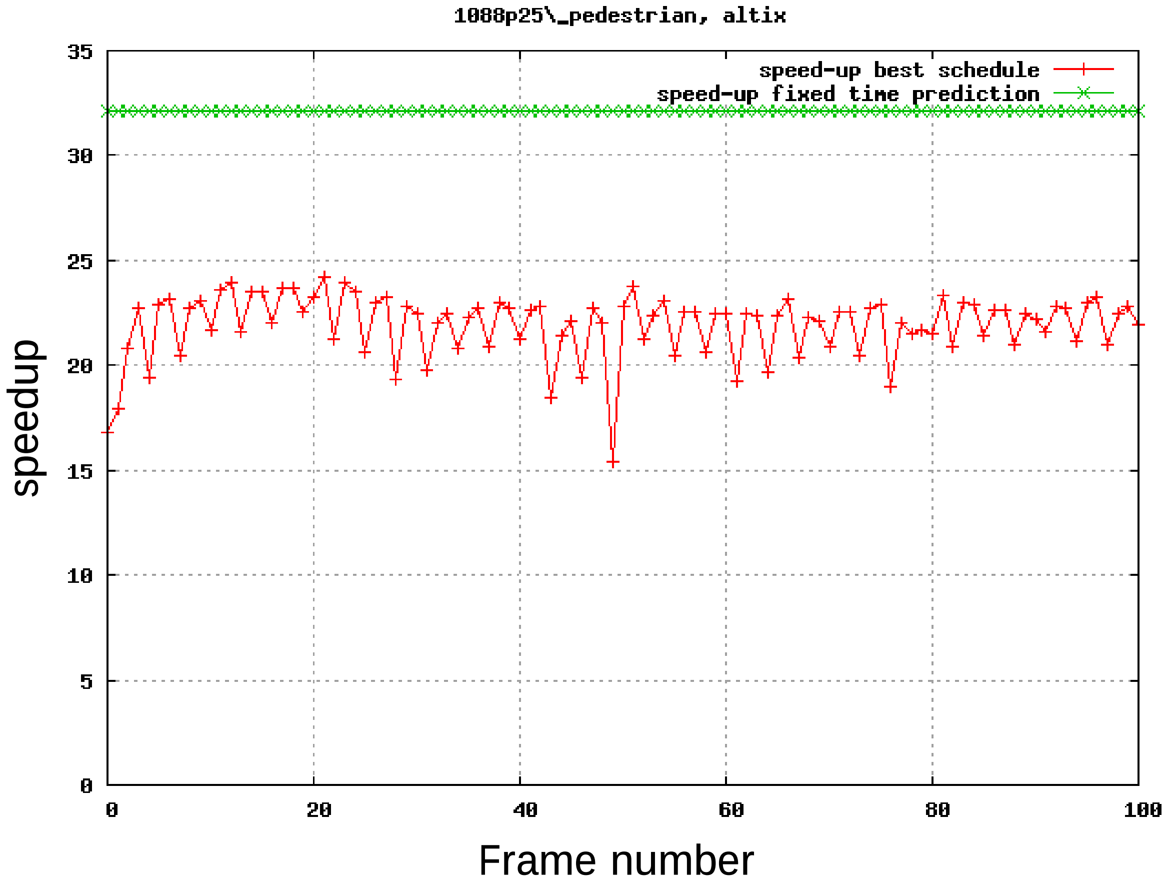
```
void deblock_mb(int x, int y)
{
again:
  ... the actual work...

  ready1 = x >= 2 && y != h &&
           tp_atomic_decrement(&deblock_ready[x-1][y+1], 1) == 0;
  ready2 = x != w && tp_atomic_decrement(&deblock_ready[x+1][y], 1) == 0;

  if(ready1 && ready2) {
    tp_submit(deblock_mb, x - 1, y + 1);   // submit left-down block
    x++;          // goto right block
    goto again;
  }
  else if(ready1) {
    x--;          // goto to left-down block
    y++;
    goto again;
  }
  else if(ready2) {
    x++;          // goto right block
    goto again;
  }
}
```

- Reduces task pool overhead
- Improves locality of reference

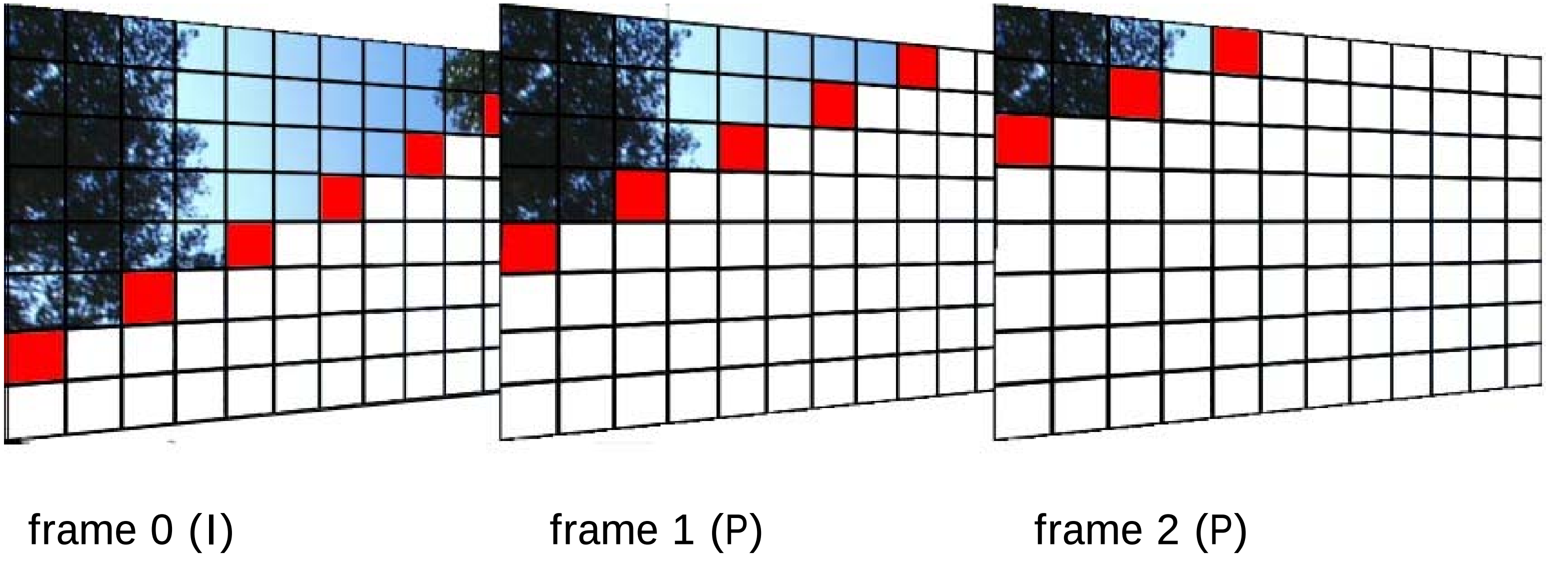
2D-Wave max scalability



- 32x for ideal conditions (constant MB decoding time)
- 23x for real video (variable MB decoding time)

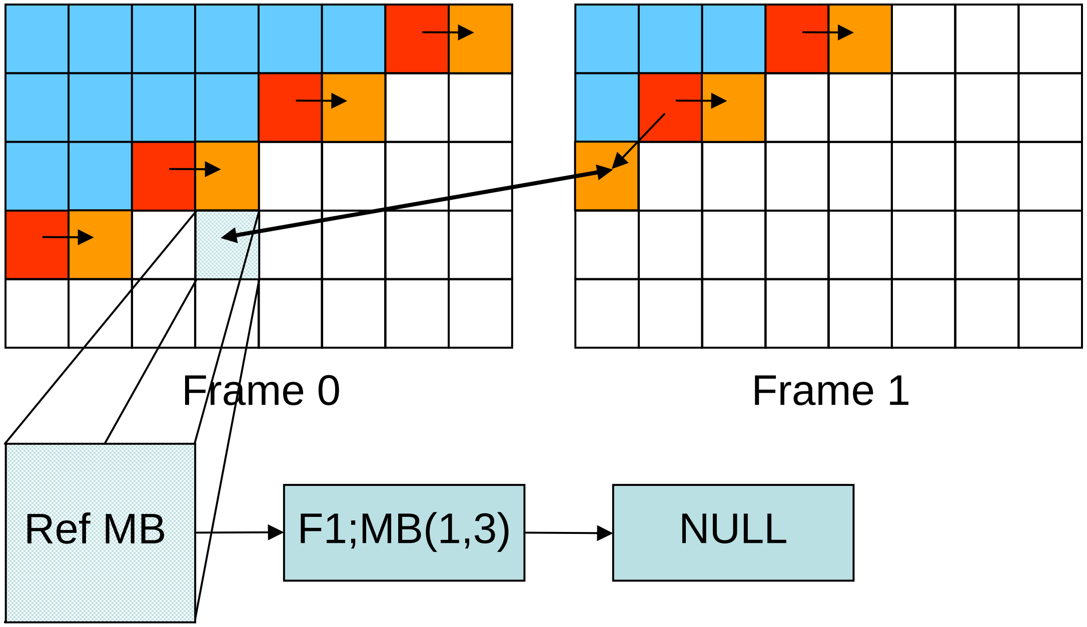
3D-Wave

- How to increase scalability?
- 3-Wave: exploit intra-frame and inter-frame MB-level parallelism
    - motion vectors typically short

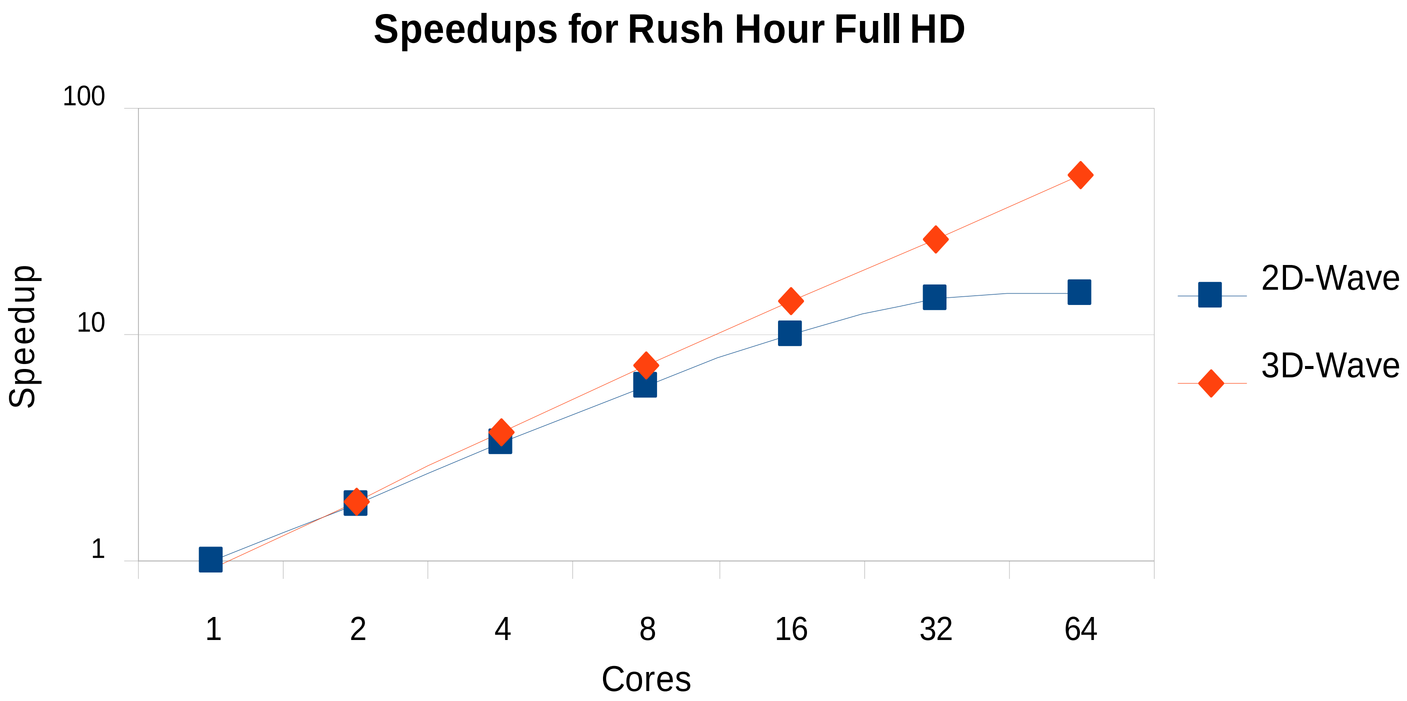


3D-Wave Implementation

- Implementation more complex than 2D-Wave due to complex, dynamic, inter-frame dependencies
  - developed a subscription mechanism where tasks subscribe themselves to a kick-off list associated with reference MB



3D-Wave Scalability



- Speedup of >51 (efficiency >80%) for 64 cores
- Start-up and end-down of short sequence (25 frames) limit efficiency
- 64 cores is 16x faster than real-time for FHD
- 3D-Wave more scalable than 2D-Wave because
  - exhibits more TLP
  - 3D-Wave spawns fewer thread due to excess TLP

Increasing Programmability

- Programming is difficult
- Parallel programming is more difficult
- Efficient parallel programming is extremely difficult

- In 2D- and 3D-Wave programmer has to take care of:
  - static task dependencies
  - dynamic task dependencies
  - optimizing data locality
  - ...

- Can we relieve the programmer from this burden?

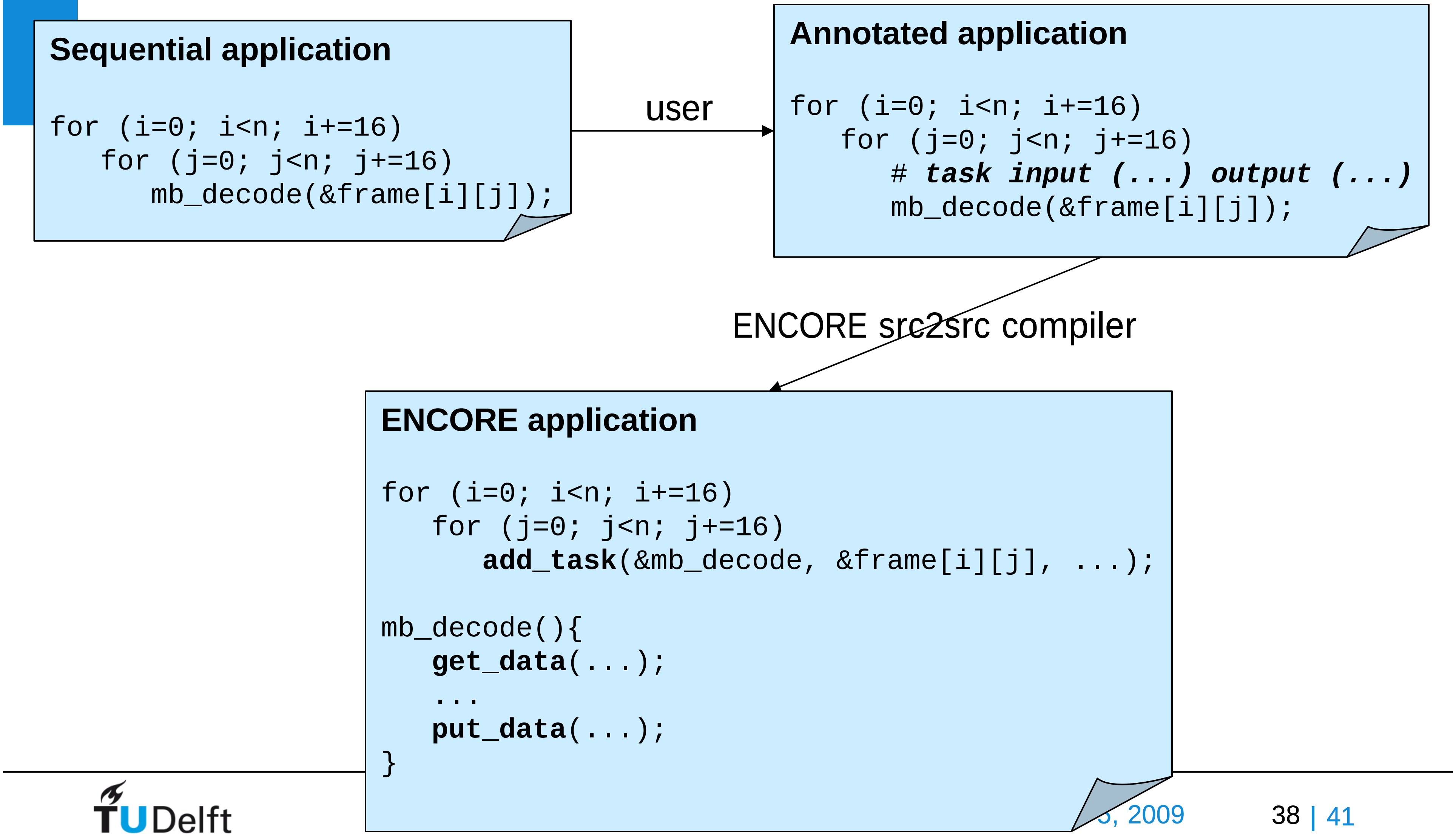
ENCORE Project

- Programmer only has to specify the tasks and the inputs and outputs of those tasks
- Runtime system takes care of
  - scheduling
  - optimizing for data locality
  - ...

- Challenges:
  - How to specify static task dependencies?
  - How to balance the workload?
  - How to specify dynamic data dependencies?
  - How to specify communication volumes?
  - How to make sure that RTS does not become a bottleneck
  - ...

ENCORE Programming Model

- Based on Open-MP



Sequential application

```
for (i=0; i<n; i+=16)
   for (j=0; j<n; j+=16)
      mb_decode(&frame[i][j]);
```

user

Annotated application

```
for (i=0; i<n; i+=16)
   for (j=0; j<n; j+=16)
      # task input (...) output (...)
      mb_decode(&frame[i][j]);
```

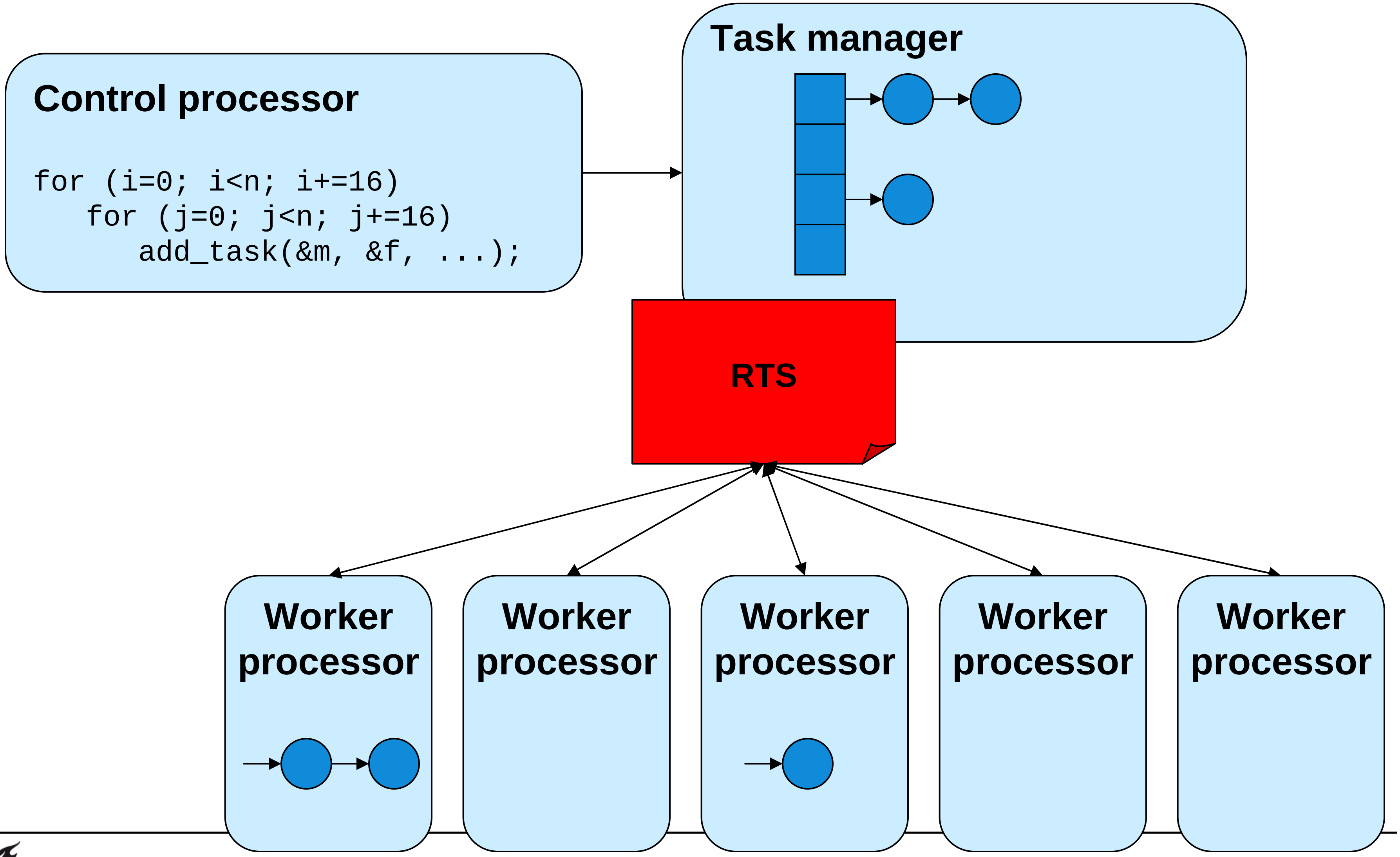
ENCORE src2src compiler

ENCORE application

```
for (i=0; i<n; i+=16)
   for (j=0; j<n; j+=16)
      add_task(&mb_decode, &frame[i][j], ...);

mb_decode(){
   get_data(...);
   ...
   put_data(...);
}
```

Encore Runtime Environment and Architecture Vision



```
for (i=0; i<n; i+=16)
   for (j=0; j<n; j+=16)
      add_task(&m, &f, ...);
```

Conclusion

- Many scheduling issues that now have to handled by expert programmers
- If parallel computing is to become a success, we have to hide (most of) the complexity

Acknowledgments

- Some slides due to or based on Jan Hoogerbrugge of NXP, Alex Ramirez and Mauricio Alvarez of BSC, Arnaldo Azevedo and Cor Meenderinck of TU Delft, . . .

- Work supported in part by Dutch organization for scientific research (NWO), EU FP6 project SARC, HiPEAC NoE, . . .