Energy vs time performances of HPC applications on Tegra K1

E. Calore, S.F. Schifano, R. Tripiccione

Enrico Calore

INFN and University of Ferrara, Italy

Laboratori Nazionali di Frascati
27th May 2015

Outline

Outline

Exploiting the NVIDIA Tegra K1 for HPC applications

Why?

- To use embedded hardware may cost less
- To use embedded hardware may consume less

Execution time for the single processor will be higher than for HPC hardware

Need to identify new metrics

- Cost per GFLOP (in Dollars/Euro):
  We need to know hardware cost
- Energy to solution (in Joule):
  We need to measure energy consumption

Exploiting the NVIDIA Tegra K1 for HPC applications

Why?

- To use embedded hardware may cost less
- To use embedded hardware may consume less

Execution time for the single processor will be higher than for HPC hardware

Need to identify new metrics

- Cost per GFLOP (in Dollars/Euro):
  We need to know hardware cost
- Energy to solution (in Joule):
  We need to measure energy consumption

Exploiting the NVIDIA Tegra K1 for HPC applications

Why?

- To use embedded hardware may cost less
- To use embedded hardware may consume less

Execution time for the single processor will be higher than for HPC hardware

Need to identify new metrics

- Cost per GFLOP (in Dollars/Euro):
  We need to know hardware cost
- Energy to solution (in Joule):
  We need to measure energy consumption

Outline

Outline

Choosing what to measure

Actually, there are several energy metrics

- Instantaneous Power consumption (Watt)
- Average Power consumption during execution (Watt)
- Energy to solution (Joule)

Sampling the instantaneous Current absorption, all the other metrics could be derived knowing the execution time T and the supply voltage V:

$$p[n] = v[n] \times i[n]$$
$$v[n] = V \ \forall \ n$$

$$N = T \times F_{\text{samp}}$$

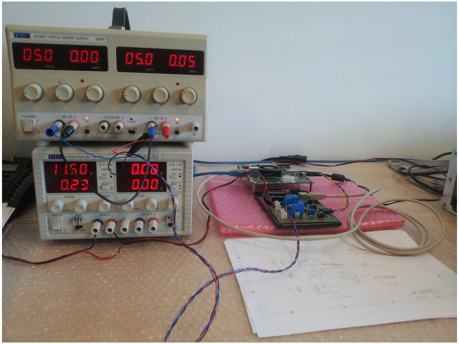
$$P_{\text{avg}} = \frac{1}{N} \sum_{n=0}^{N-1} p[n]$$

$$E_{\text{tosol}} = \frac{1}{F_{\text{samp}}} \sum_{n=0}^{N-1} p[n]$$

Setup to sample instantaneous current absorption

Shopping list

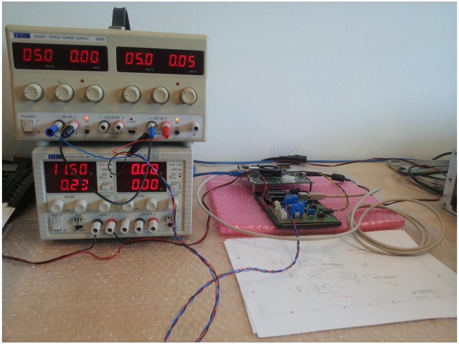
- A USB/GPIB/RS232 digital power meter
- One current to voltage converter plus
  an Arduino UNO (microcontroller + 10-bit ADC + Serial over USB)



Setup to sample instantaneous current absorption

Shopping list

- A USB/GPIB/RS232 digital power meter
- One current to voltage converter plus
  an Arduino UNO (microcontroller + 10-bit ADC + Serial over USB)



Current to Voltage + Digitization with Arduino + USB Serial

Outline

Arduino Sketch code

The Arduino board waits a char on the serial connection over the USB port.
'A': starts to digitize at 1kHz, storing samples in its memory till it runs out of it
'S': sends the acquired samples over the Serial connection emptying its buffer

```
//SerialEvent occurs whenever a new
//data comes in the hardware serial RX.
void serialEvent() {
  while (Serial.available()) {
    // get the new byte:
    char inChar = (char)Serial.read();
    // Send buffer via Serial
    if (inChar == 'S') {
      acquireData = false;
      sendData = true;
    // Start data acquisition
    } else if (inChar == 'A') {
      acquireData = true;
      sendData = false;
      idx = 0;
    }
  }
}
```

```
// This is called every ms
ISR(TIMER0_COMPA_vect) {
 if (acquireData) {
   byte i;
   unsigned int sensorValue = 0;
   // Average over avgSamples readings
   // one read costs about 0.11ms
   for (i = 0; i < avgSamples; i++) {
     // read the input on analog pin0
     sensorValue += analogRead(A0);
   }
   isendBuffer[idx] = sensorValue;
   idx++;
 }
}
```

From the host code

To measure the energy consumption of a specific function, we can start the acquisition just before starting the computations:

```
int fd;
struct termios newtio, oldtio;
char filename[256]; // filename were to store acquired data

// Initialize Arduino Serial connection
fd = init_serial(&oldtio, &newtio);

// Start arduino data acquisition
start_arduino_acq(fd);
usleep(10000); // Wait a bit to have some baseline points in the plot

run_my_function();

// Start arduino data read-out
start_arduino_readout(fd, filename, 900);
close_serial(fd, &oldtio);
```

To store acquired data in the Arduino memory grants for minimal interferences with the code execution in the Jetson board.

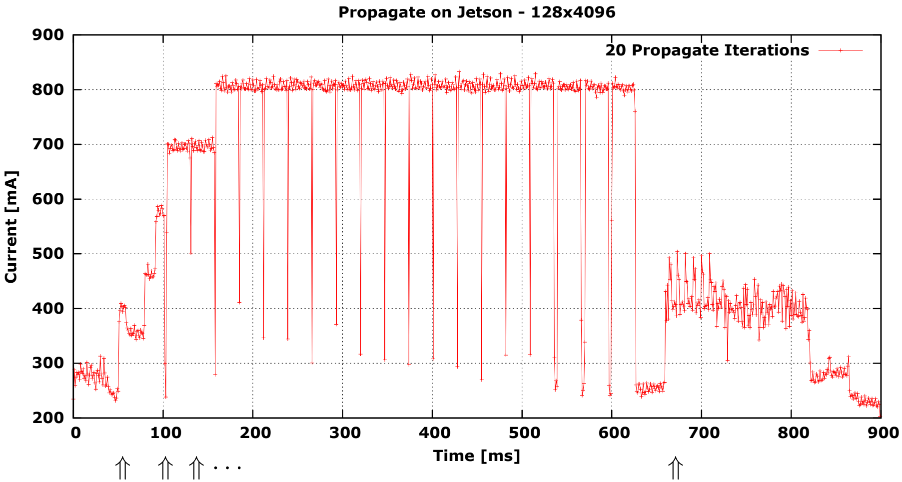
From the host code

To measure the energy consumption of a specific function, we can start the acquisition just before starting the computations:

```
int fd;
struct termios newtio, oldtio;
char filename[256]; // filename were to store acquired data

// Initialize Arduino Serial connection
fd = init_serial(&oldtio, &newtio);

// Start arduino data acquisition
start_arduino_acq(fd);
usleep(10000); // Wait a bit to have some baseline points in the plot

run_my_function();

// Start arduino data read-out
start_arduino_readout(fd, filename, 900);
close_serial(fd, &oldtio);
```

To store acquired data in the Arduino memory grants for minimal interferences with the code execution in the Jetson board.

Acquired data example with default frequency scaling



Iterations can be counted

This is a D2H transfer

Outline

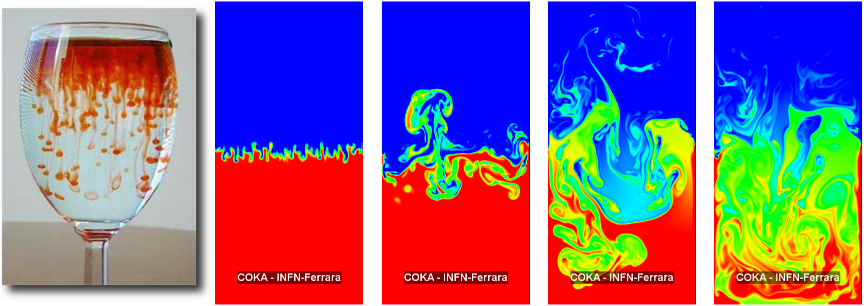
The D2Q37 Lattice Boltzmann Model

- Lattice Boltzmann method (LBM) is a class of computational fluid dynamics (CFD) methods
- simulation of synthetic dynamics described by the discrete **Boltzmann** equation, instead of the **Navier-Stokes** equations
- a set of **virtual particles** called **populations** arranged at edges of a discrete and regular grid
- interacting by **propagation** and **collision** reproduce – after appropriate averaging – the dynamics of fluids
- D2Q37 is a D2 model with 37 components of velocity (populations)
- suitable to study behaviour of **compressible** gas and fluids optionally in presence of **combustion** [1] effects
- correct treatment of Navier-Stokes, heat transport and perfect-gas ($P = \rho T$) equations

[1] chemical reactions turning cold-mixture of reactants into hot-mixture of burnt product.

Simulation of the Rayleigh-Taylor (RT) Instability

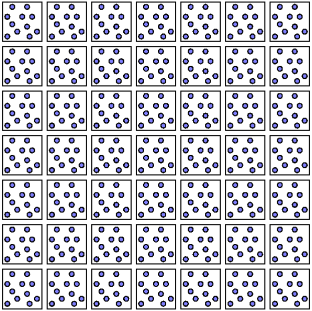
Instability at the interface of two fluids of different densities triggered by gravity.



A cold-dense fluid over a less dense and warmer fluid triggers an instability that mixes the two fluid-regions (till equilibrium is reached).

Computational Scheme of LBM

```
foreach time-step

  foreach lattice-point
    propagate();
  endfor

  foreach lattice-point
    collide();
  endfor

endfor
```



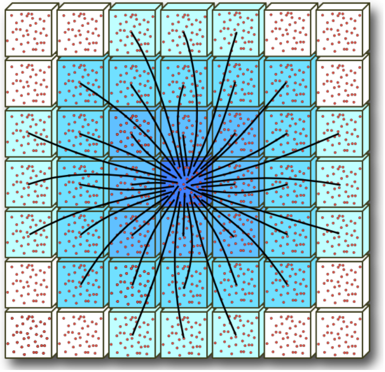
Embarassing parallelism

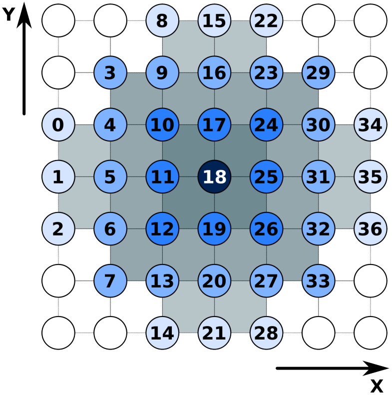
All sites can be processed in parallel applying in sequence propagate and collide.

Challenge

Design an efficient implementation able exploit a large fraction of available peak performance.

D2Q37: propagation scheme





- perform accesses to neighbour-cells at distance 1,2, and 3
- generate memory-accesses with **sparse** addressing patterns

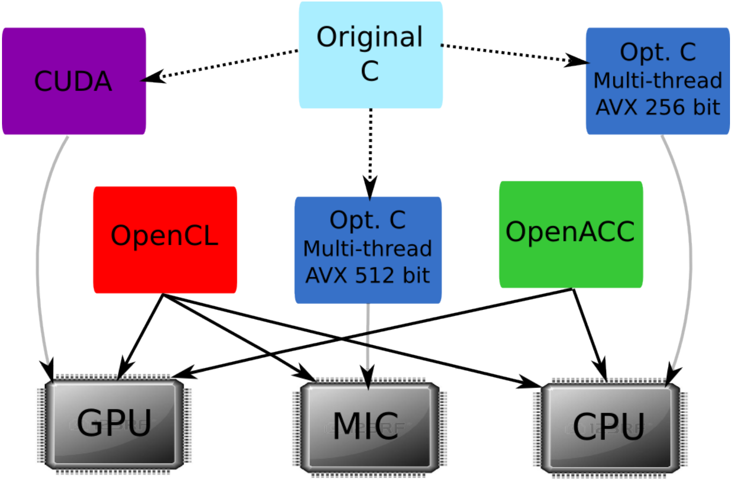
D2Q37 collision

- collision is computed at each lattice-cell after computation of boundary conditions
- computational intensive: for the D2Q37 model requires $\approx$ 7500 DP floating-point operations
- completely local: arithmetic operations require only the populations associate to the site

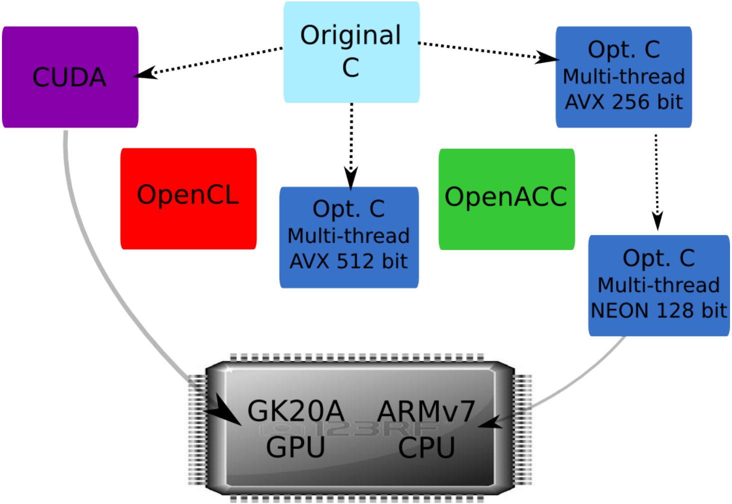
- computation of propagate and collide kernels are kept separate
- after propagate but before collide we may need to perform collective operations (e.g. divergence of of the velocity field) if we include computations conbustion effects.

Outline

Initial Code implementations



Code implementations



Outline

1. Introduction
2. Measuring the energy consumption
   - How to measure
   - Managing the acquisition
3. Lattice Boltzmann Model (D2Q37)
   - Code Implementations
   - Managing the Frequency Scaling
   - C with NEON intrinsics, on the Cortex A15
   - CUDA on the GK20A
4. Conclusions

Manually setting the Jetson clocks

Manually setting CPU frequency and online cores (LP vs G)

```
echo "userspace" > /sys/devices/system/cpu/cpu0/cpufreq/scaling_governor
echo <frequency> > /sys/devices/system/cpu/cpu0/cpufreq/scaling_setspeed
```

```
echo 0 > /sys/devices/system/cpu/cpuquiet/tegra_cpuquiet/enable
echo 0 > /sys/devices/system/cpu/cpu1/online
echo 0 > /sys/devices/system/cpu/cpu2/online
echo 0 > /sys/devices/system/cpu/cpu3/online
echo LP > /sys/kernel/cluster/active
```

Manually setting GPU (and MEM) frequency

```
cat /sys/kernel/debug/clock/gbus/possible_rates
72000 108000 180000 252000 324000 396000 468000 540000
612000 648000 684000 708000 756000 804000 852000 (kHz)

echo 852000000 > /sys/kernel/debug/clock/override.gbus/rate
echo 1 > /sys/kernel/debug/clock/override.gbus/state
```

Optimizing for the new metrics

Using the regular time to solution metric (aka faster is better)

- Highest CPU frequency is the best
- Highest GPU frequency is the best
- Highest MEM frequency is the best
- Best software parameters have to be identified (e.g. CUDA block size)

Using the energy to solution metric

- Which CPU frequency is the best?
- Which GPU frequency is the best?
- Which MEM frequency is the best?
- Best software parameters have still to be identified (e.g. CUDA block size)

Optimizing for the new metrics

Using the regular time to solution metric (aka faster is better)

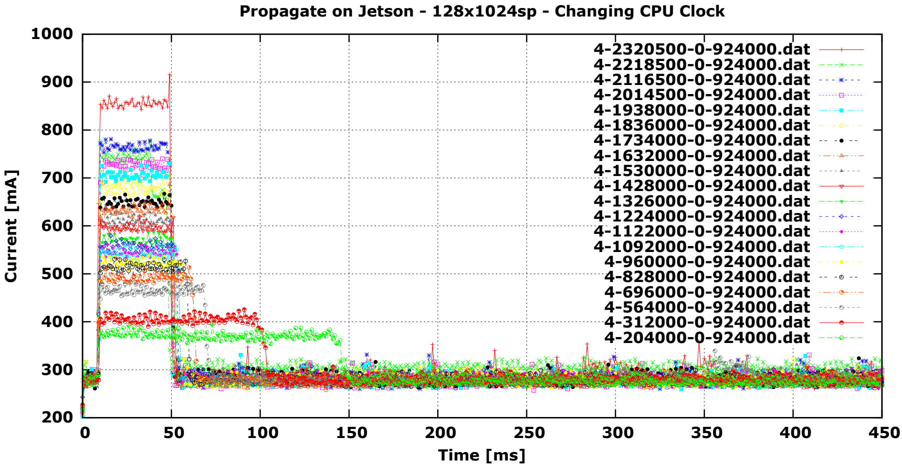
- Highest CPU frequency is the best
- Highest GPU frequency is the best
- Highest MEM frequency is the best
- Best software parameters have to be identified (e.g. CUDA block size)

Using the energy to solution metric

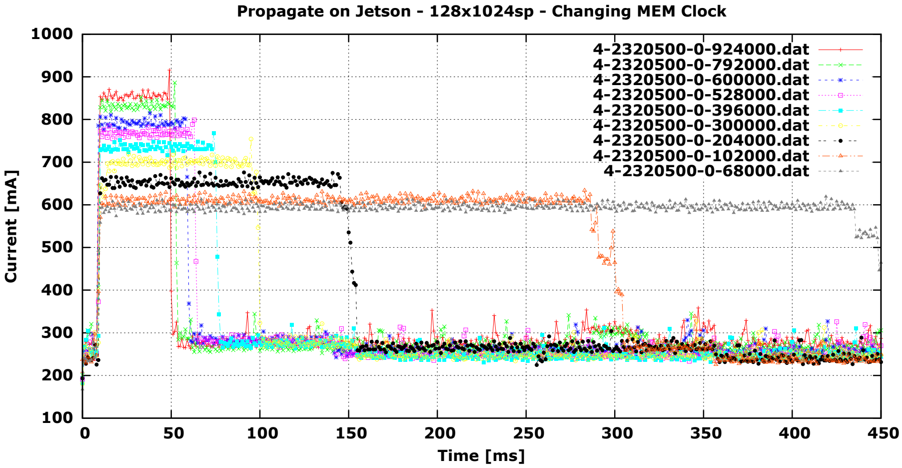
- Which CPU frequency is the best?
- Which GPU frequency is the best?
- Which MEM frequency is the best?
- Best software parameters have still to be identified (e.g. CUDA block size)

Outline

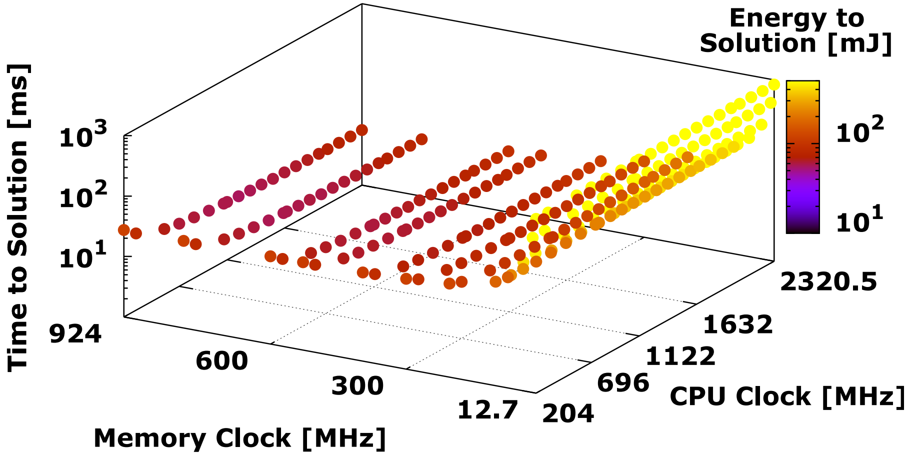
Propagate changing the G cluster clock



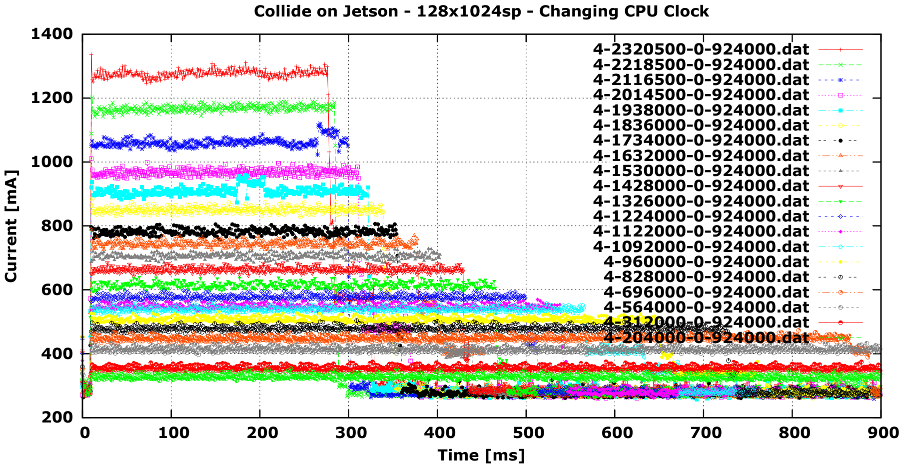
Propagate changing the MEM clock



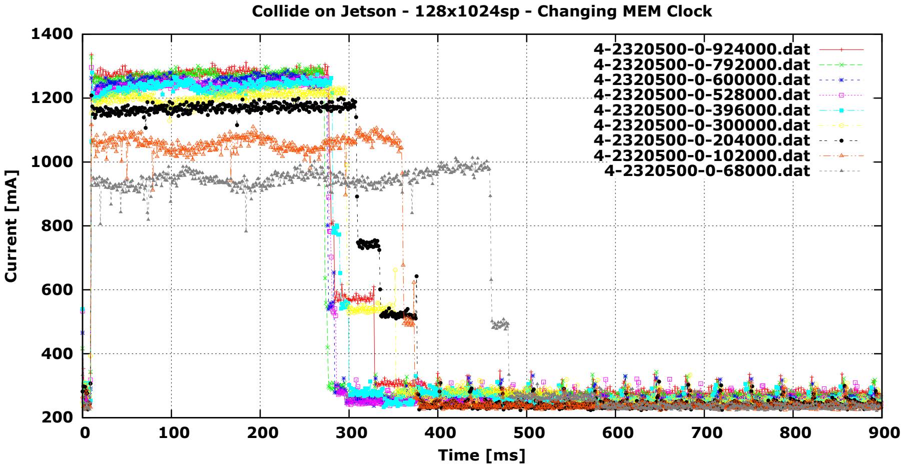
Time and Energy to solution (Propagate)



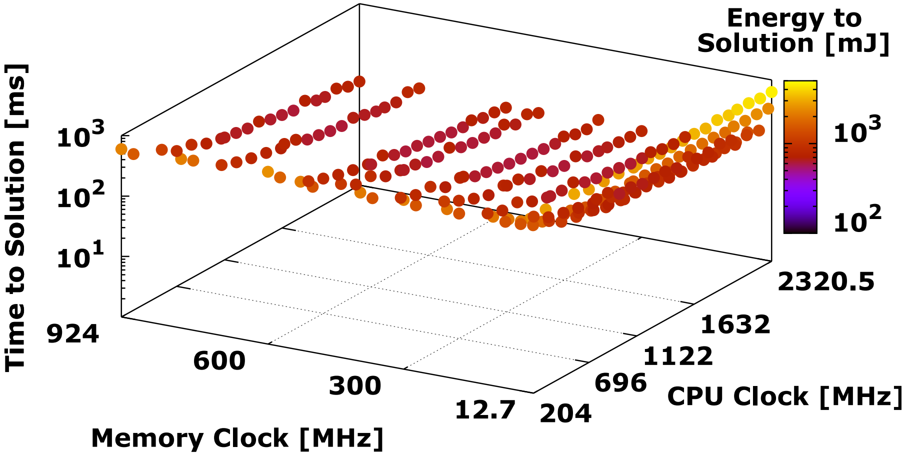
Collide changing the G cluster clock



Collide changing the MEM clock

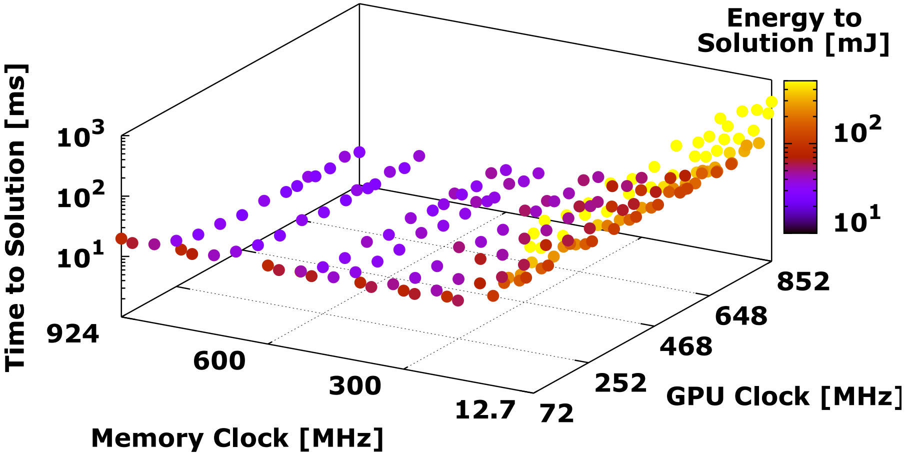


Time and Energy to solution (Collide)

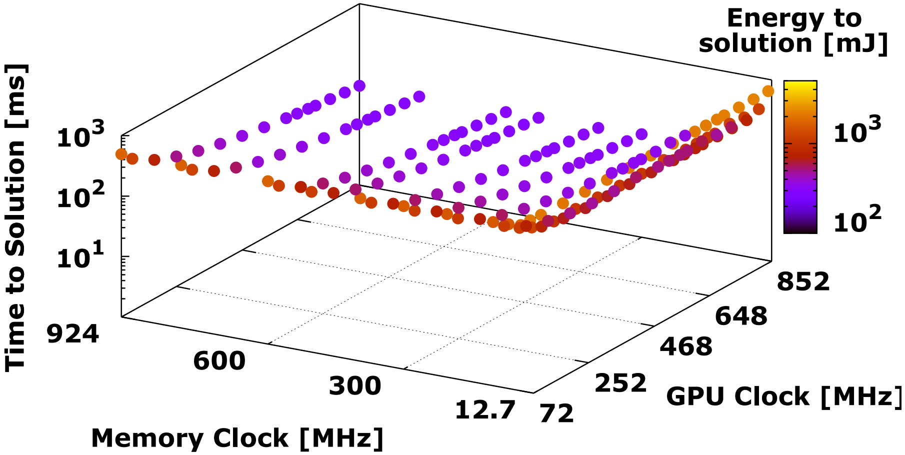


Outline

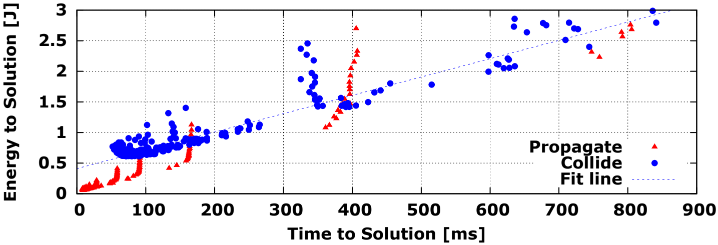
Time and Energy to solution (Propagate)

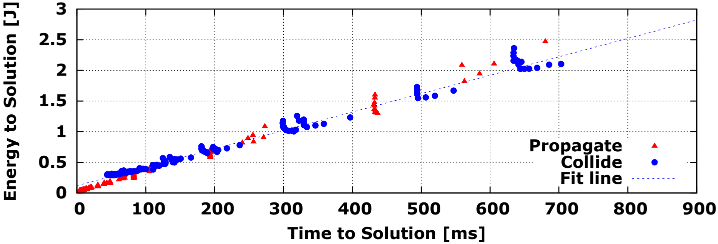


Time and Energy to solution (Collide)

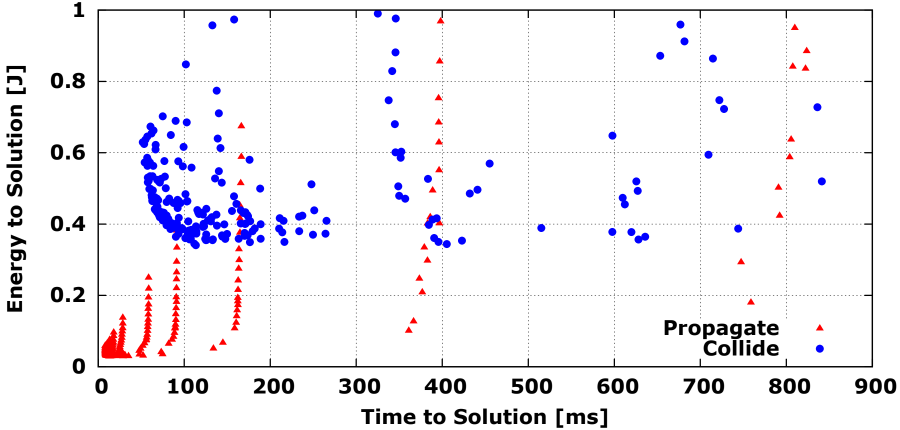


Energy to Sol. vs Time to Sol. CPU(top), GPU(bottom)

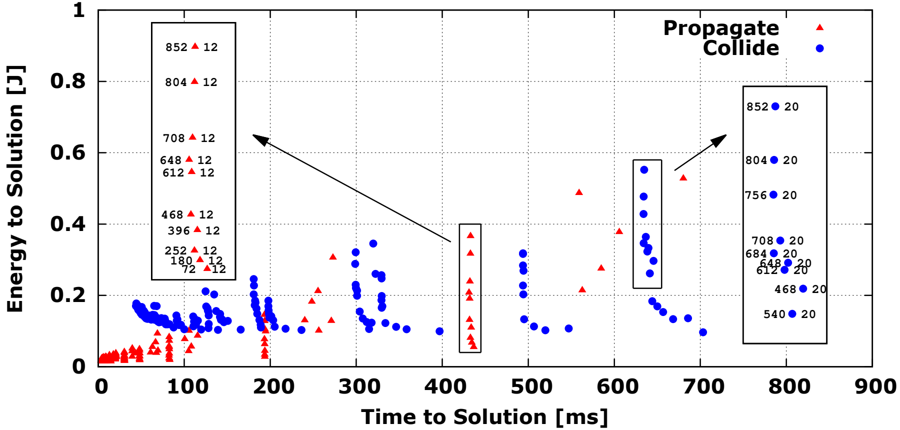




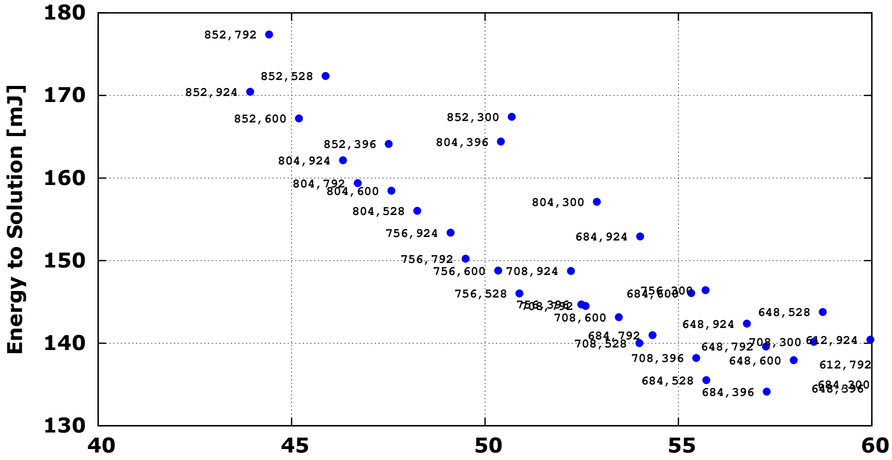
Energy to Solution vs Time to Solution (CPU)



Energy to Solution vs Time to Solution (GPU)



Energy to Solution vs Time to Solution (GPU) zoom



Outline

Conclusions

- baseline power consumption (leakage current + ancillary electronics) is relevant concerning the whole energy budget.
- limited but not negligible power optimization is possible by adjusting clocks on a kernel-by-kernel basis ($\approx 20 \cdots 30\%$).
- best region is close to the system highest frequencies.
- options to run the processor at very low frequencies seem almost useless; if possible would be interesting to be able to remove power from the (sub-)system while idle.

Future works

- perform similar measurements on a high-end node and compare results.
- test on newer low-power processors, such us the Tegra X1.
- consider not only hardware-based tuning, but also software tuning.

Conclusions

- baseline power consumption (leakage current + ancillary electronics) is relevant concerning the whole energy budget.
- limited but not negligible power optimization is possible by adjusting clocks on a kernel-by-kernel basis ($\approx 20 \cdots 30\%$).
- best region is close to the system highest frequencies.
- options to run the processor at very low frequencies seem almost useless; if possible would be interesting to be able to remove power from the (sub-)system while idle.

Future works

- perform similar measurements on a high-end node and compare results.
- test on newer low-power processors, such us the Tegra X1.
- consider not only hardware-based tuning, but also software tuning.

Thanks for Your attention