Mathematica Parallel Computing. Some Timing Results on a Intel Nehalem Multicore Processor

Béla Paláncz[1], Levente Kovács [2]

[1] Department of Photogrammetry and Geoinformatics, Faculty of Civil Engineering, Budapest University of Technology and Economics, 1111 Budapest, Műegyetem rkp. 3, Hungary; palancz@epito.bme.hu

[2] Department of Control Engineering and Information Technology, Faculty of Electrical Engineering and Informatics, Budapest University of Technology and Economics, H-1117 Budapest, Magyar Tudósok krt. 2, Hungary; lkovacs@iit.bme.hu

Abstract: Some frequently employed algorithms in engineering, are parallel by nature (embarrasingly parallel algorithms) and some others can be parallelized via data parallelization. Algorithms like probability analysis, linear homotopy continuation method, Gauss-Jacobi combinatorial technique are belonging to the first group, while others like algorithms for digital image processing as well as reduced Groenber basis application to solving systems of polynomial equations fall into the other category. In this case study we illustrate how Mathematica can manage to evaluate such algorithms parallel on a multicore machine. The analysis of the efficiency of the computation and the net reduction of the execution time are presented by three examples as well as some useful tips are given to avoid pitfalls and utilize the advantages of parallel processing.

Keywords: parallel computation, multicore processor, Monte-Carlo method, Gauss-Jacobi Algorithm, color quantization.

1 Introduction

On one hand, widespreading of multicore PC's in the market provides ample proof that we are in the multicore era. In years to come probably the number of cores packed into a single chip will represent the increasing computational power instead of the clock frequency of the processor, which has reached its limit and will likely stay below 4 GHz for a while.

On the other hand, popular computational software systems like MATLAB and *Mathematica* extended their codes with simple instructions and functions collected

in a toolbox or in an application providing a comfortable access and realization of parallel computations on multicore desktop or even laptop computers.

Consequently, parallel computing can be utilized by not only experts but engineers and scientists having no special knowledge in this area. However, despite of the easy accesable hardwares and the simple user friendly softwares, there are some pitfalls and tricks, which good to know in order to avoid unexpected negative effects and problems as well as to be able to exploit the advantages of a multicore system.

Using illustrative examples, important features of parallel computation will be demonstrated and show how some frequently employed algorithms can be efficiently evaluated in parallel.

1.1 Implicit and Explicit Parallelism

Multicore processors is a processing system composed of two or more independent cores. The amount of performance gained by the use of a multicore processor is strongly depended on the software algorithms and implementation. In particular, the possible gains are limited by the fraction of the software that can be "parallelized" to run on multiple cores simultaneously; this effect is described by Amdahl's law. In the best case, so-called embarrasingly parallel problems may realize speedup factors near the number of cores. Many typical applications, however, do not realize such large speedup factors, and thus the parallelization of software is a significant on-going topic of research.

On multicore machine so many indpendent processes can be executed in parallel as many cores the machine has. However using so called multithreading execution model allowing to realize further parallelization (since within the context of a single process running on one core) more concurrently tasks sharing same resources can be executed.

For example, we used Nehalem i7 (Bloomfield) 940 processor, which has four cores and using simultaneous multithreading enabling two threads per core [1].

Some of the programming languages are able to exploit multicore and multithreading ability, automatically, partly or fully, without any special directives of the programming language. This characteristic of a programming language is called *implicit parallelism*. *Mathematica* also supports automatically the parallelization of some operations, but in modest way, althoug already Version 5.2 (2005) added automatic multi-threading when computations are performed on multi-core computers.

Paralleism provided by the code extention and controlled by the user, namely using or not using parallel execution is called *explicit parallelism*. A feature of a programming language for a parallel processing system allows or forces the

programmer to annotate his program indicating which parts should be executed as independent parallel tasks. This is obviously more work for the programmer than a system with implicit parallelism (where the system decides automatically which parts to run in parallel) but may allow higher performance. *Mathematica* has many functions supporting explicit parallelism and we will use some of them in the following examples.

2 Task Parallel

The type of parallel jobs can be task parallel (embarrassingly parallel) and data parallel. In case of the task parallel, multiple workers work on different parts of the problem without communication between each others. Therefore, the result will be independent of the execution order. In the followings we demonstrate this type of evaluation by two examples.

2.1 Ex. 1 - Infinite Slope Stability via Monte- Carlo Analysis

We are going to compute the safety factor for an infinite slope subjected to given ground accelaration value occuring over a specified time period [2]. The safety factor μ, is the ratio of resisting forces to driving forces. A slope will be stable if the resisting forces exceed the driving forces and the factor of safety is greater than 1. We will use a simplified model of the infinite slope model. Let φ be the angle of internal friction (representing the frictional component of soil shear strength), β be the slope angle in degrees, and $0 \leq H \leq 1$ is the dimensionless height of the prheatic surface above the base of the slide mass. The pseudostatic factor of safety of an infinite slope subjected to seismic acceleration is then,

$$\mathbf{SeismicsFS}[\varphi\_, \beta\_, H\_, Cs\_] := \frac{((1 - H/2)\,\mathrm{Cos}[\beta] - Cs\,\mathrm{Sin}[\beta])\,\mathrm{Tan}[\phi]}{\mathrm{Sin}[\beta] + Cs\,\mathrm{Cos}[\beta]} \qquad (1)$$

in which *Cs* is a coefficient of seismic acceleration given in terms of the gravitational acceleration *g*. This is a generalization of the static model (*Cs* = 0).

According to the USGS national earthquake hazard maps, a peak ground acceleration of 0.12 *g* has 0.10 probability of being exceeded in 50 years in Socorro, New Mexico. So we consider the following *Mathematica* function for computing the safety factor at a fixed Cs = 0.12 value,

```
Slope[n_] := Module[{φ, β, H, result, i, s, Cs=0.12}, result={};
  Do[φ = Random[UniformDistribution[{30.°,35.°}]];
     β = Random[UniformDistribution[{20.°,25.°}]];
```

H = **Random**[**LogNormalDistribution**[-2.31,1.33]];

s = **SeismicsFS**[φ, β,H,Cs];

If[s ≥ 0, **AppendTo**[result, s]],{i, n}]; result]

We will censor the offensive negative values by simply removing them. Let us consider 80 000 trials. The result can be seen in Figure 1. The computation time in this case is 27.703 sec.

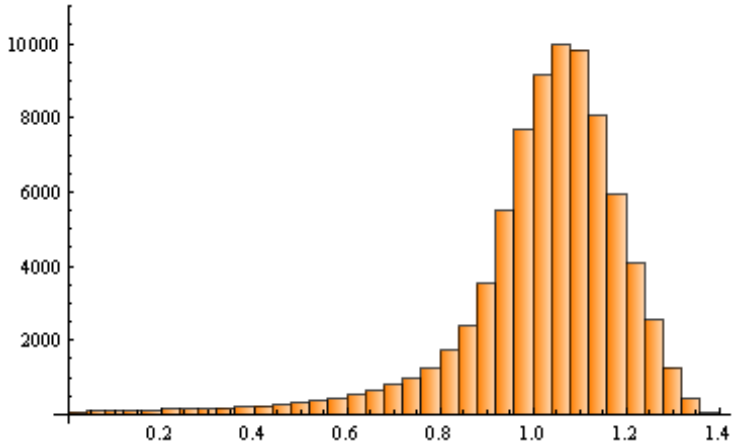

Figure 1

The histogram of the distribution of the safety factor in case Cs = 0.12, employing 80 000 samples

Realising the same scenario in parallel way (having eight threads, Figure 2)

DistributeDefinitions[**Slope**, **SeismicsFS**]

n = **Table**[10000,{8}]

μ = **Flatten**[**ParallelMap**[**Slope**[#] & ,n]

the computaion time is considerably decreased with result obtained in 2.7812 sec.

We have checked the example for the other in parallel evaluation method too,

n = **Table**[10000,{8}]

DistributeDefinitions[**Slope**, **SeismicsFS**, n]

μ = **Flatten**[**WaitAll**[**Map**[**ParallelSubmit**[**Slope**[#]]& , n]]]

obtaining nearly the same computation time 2.7812 sec.

The preformance metrics are summarized in Table 1.

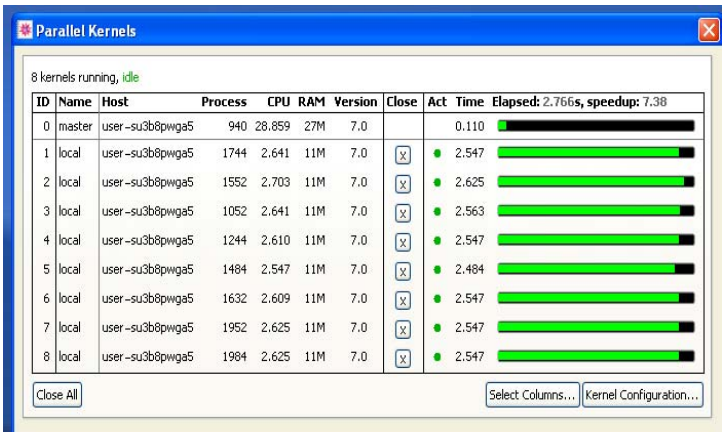

Figure 2
Status of the kernels in case of parallel Monte- Carlo simulation with the infinite slope model using **ParallelMap**.

Table 1
Performance of different types of the computations of the Monte- Carlo simulation with the infinite slope model.

Comput. type	Time (sec.)	Speedup	Efficiency (%)	Relative unbalance	Mean value of μ	Standard deviation
Non-parallel	27.70	-	12.5	-	1.02153	0.180758
ParallelMap	2.78	7.38	124.6	0.06	1.01886	0.183263
ParallelSubmit	2.78	7.40	124.6	0.07	1.01827	0.183293

2.2 Ex. 2 - Photogrammetric Positioning by Gauss- Jacobi Algorithm

The relation between the coordinates of a ground point (X_i, Y_i, Z_i) and the coordinates of the corresponding point on the photo plain (x_i, y_i) can be represented by the following linear transformation (Figure 3),

$$\begin{pmatrix} x_i - \eta_0 \\ y_i - \xi_0 \\ -f \end{pmatrix} = k_i R \begin{pmatrix} X_i - X_0 \\ Y_i - Y_0 \\ Z_i - Z_0 \end{pmatrix} \tag{2}$$

where

- η_0, ξ_0 are the coordinates of the perspective center on the photo plane;
- f the focal length;
- k_i the scaling factor;
- R the rotation matrix;
- X_0, Y_0, Z_0 the coordinates of the perspective center in the ground system.

In general the focal length f is known, and the parameters of the transformation should be determined on the basis of the coordinates of 3 corresponding plane-ground points.

The problem with this representation of the transnformation is, that the scaling factor is different in the different points, therefore it is reasonable to eliminate it. Let us express the equations of the transformation for the three different coordinates,

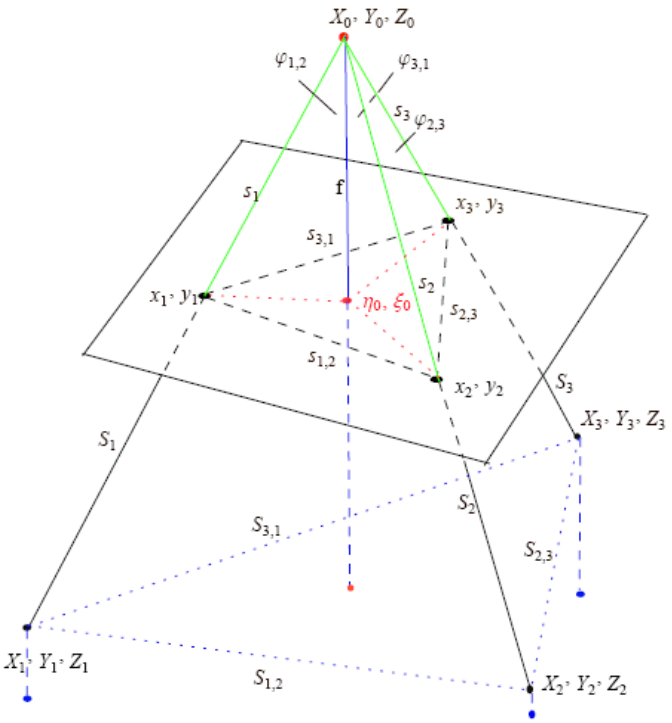

Figure 3
Photogrammetric 3D resection.

$$
\begin{aligned}
x_i - \eta_0 &= k_i (R_{1,1}(X_i - X_0) + R_{1,2}(Y_i - Y_0) + R_{1,3}(Z_i - Z_0)) \\
y_i - \xi_0 &= k_i (R_{2,1}(X_i - X_0) + R_{2,2}(Y_i - Y_0) + R_{2,3}(Z_i - Z_0)) \\
-f &= k_i (R_{3,1}(X_i - X_0) + R_{3,2}(Y_i - Y_0) + R_{3,3}(Z_i - Z_0))
\end{aligned} \tag{3}
$$

Dividing the first and second equation by the third one and rearrange them, introducing $r_{ij} = R_{i,j}$ we get

$$
\begin{aligned}
x_i - \eta_0 &= -f \frac{r_{11}(X_i - X_0) + r_{12}(Y_i - Y_0) + r_{13}(Z_i - Z_0)}{r_{31}(X_i - X_0) + r_{32}(Y_i - Y_0) + r_{33}(Z_i - Z_0)} \\
y_i - \xi_0 &= -f \frac{r_{21}(X_i - X_0) + r_{22}(Y_i - Y_0) + r_{23}(Z_i - Z_0)}{r_{31}(X_i - X_0) + r_{32}(Y_i - Y_0) + r_{33}(Z_i - Z_0)}
\end{aligned} \tag{4}
$$

Now, we express the elements of the rotation matrix with the elements of the skew matrix,

$$
S = \begin{pmatrix} 0 & -c & b \\ c & 0 & -a \\ -b & a & 0 \end{pmatrix} \tag{5}
$$

The rotation matrix (where I_3 is a 3x3 identity matrix) is

$$
R = (I_3 - S)^{-1}(I_3 + S) \tag{6}
$$

As a result, R becomes

$$
R = \begin{matrix} \frac{1 + a^2 - b^2 - c^2}{1 + a^2 + b^2 + c^2} & \frac{2ab - 2c}{1 + a^2 + b^2 + c^2} & \frac{2(b + ac)}{1 + a^2 + b^2 + c^2} \\ \frac{2(ab + c)}{1 + a^2 + b^2 + c^2} & \frac{1 - a^2 + b^2 - c^2}{1 + a^2 + b^2 + c^2} & -\frac{2(a - bc)}{1 + a^2 + b^2 + c^2} \\ \frac{2(-b + ac)}{1 + a^2 + b^2 + c^2} & \frac{2(a + bc)}{1 + a^2 + b^2 + c^2} & \frac{1 - a^2 - b^2 + c^2}{1 + a^2 + b^2 + c^2} \end{matrix} \tag{7}
$$

In our case given coordinate values of the ground points and image coordinates are in Table2 and f = 153000. It means, we have 12 equations, but only 8 unknown variables (a, b, c, X_0, Y_0, Z_0, η_0, ξ_0). Consequently, our system is overdetermined.

Let us employ the Gauss- Jacobi combinatorial algorithm. We have 6 corresponding points and every 4 points represent 4 ×2 equations , see Eqs. (4). The number of the combinations is 15, so we should solve 15 subsets of eight equations.

Table 2
Coordinate values of the ground points and image coordinates [3]

X_i	Y_i	Z_i	x_i	y_i
-460	-920	-153	18996.171	-64147.679
460	-920	0	113471.749	-73694.266
-460	0	0	16504.609	16331.649
460	0	153	128830.826	21085.172
-460	920	-153	13716.588	106386.802
460	920	0	120577.473	128214.823

The solution of the subsets takes 122.85 sec, therefore it is reasonable to employ parallel computation. Defining all variables involved in the computation as parallel variables, the parallel computation can be applied straight forward (here the application of the function **Parallelize** is recommended).

As a result a considerably shorter running time, 34.54 sec. is obtained (Figure 4). It can be seen, that only 7 threads were activated and their tasks needed different running time. Indeed, the solutions have different types and different lenghts.

The performance metrics are summarized in Table 3. The relative unbalance is based on the 7 active threads.

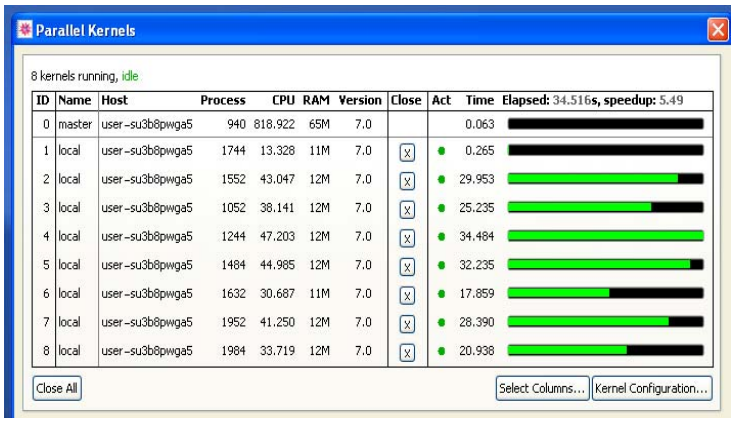

Figure 4

Status of the kernels in case of parallel evaluation of the Gauss- Jacobi algorithm using **Parallelize**

Table 3. Performance of computation of the Gauss- Jacobi combinatorial algorithm.

Comput. type	Time (sec.)	Speedup	Efficiency (%)	Relative unbalance
Non-parallel	122.86	-	12.5	-
Parallelize	34.55	5.49	44.45	0.48

3 Data Parallel

Typically in this case data being analysed is too large for one computer (one core). In these cases each worker operates on part of the data and they may or may not communicate with each other. As illustration let us consider an example of digital image processing.

3.1 Ex. 3 - Reducing Colors via Color Approximation

On systems with 24 - bit color displays, truecolor images can display up to 16 million (i.e. 2^{24}) colors. On systems with lower screen bit depths, truecolor images are still displayed reasonably well, using color approximation. Color approximation is the process by which the software chooses replacement colors in the event where direct matches cannot be found. One of the methods to carry out such color approximation is the color quantization [4].

An important term in discussions of image quantization is RGB color cube. The RGB color cube is a 3D array of all of the colors that are defined for a particular data type (Figure 5). Quantization involves dividing the RGB color cube into a number of smaller boxes, and then mapping all colors that fall within each box to the color value at the center of that box.

If the actual image is big in size, one may divide the image into parts, and this quantization process can be carried out parallel on these image segments.

Let us consider an airborne digital photo of Boston (Figure 6), a 4481 x 2881 size image, with 12 909 761 pixels.

Let us divide the RGB cube into 10 subcubes, and carry out the color reduction by color quantization using **ColorQuantize**[pBoston,10]. The computation time obtained is 23.187 sec.

Now let us divide the picture into eight parts (4481/4 = 1120 and 2881/2 = 1140)

pP = **ImagePartition**[pBoston, {1120,1440}]

see, Figure 7.

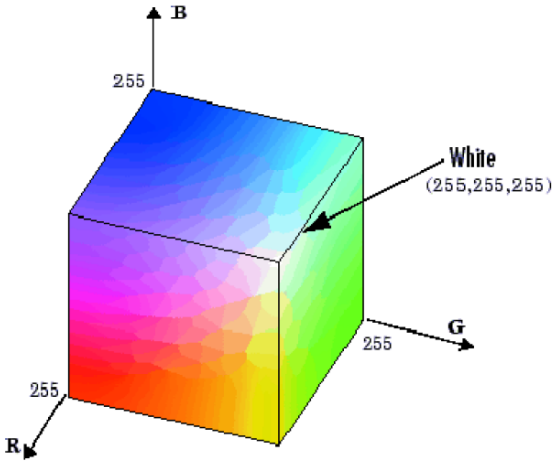

Figure 5
RGB color cube.

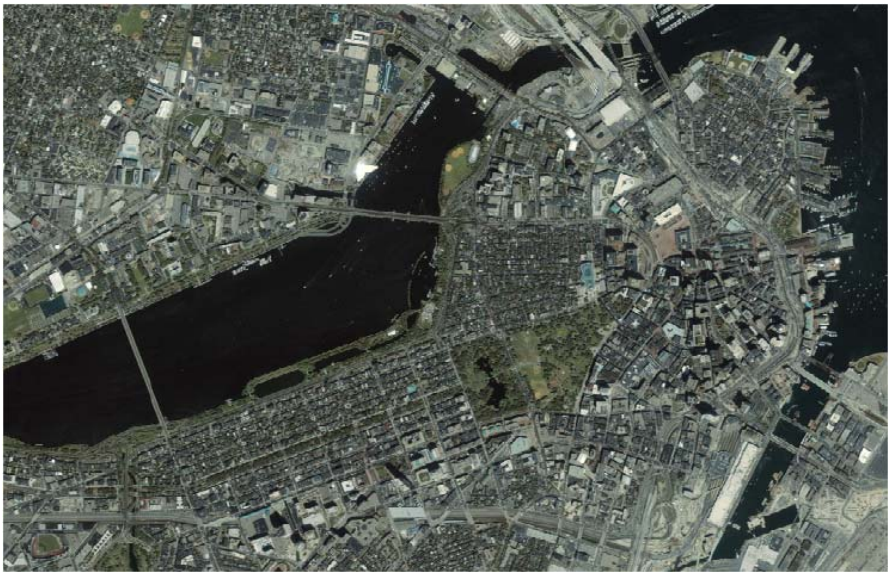

Figure 6
The considered airborne digital photo of Boston.

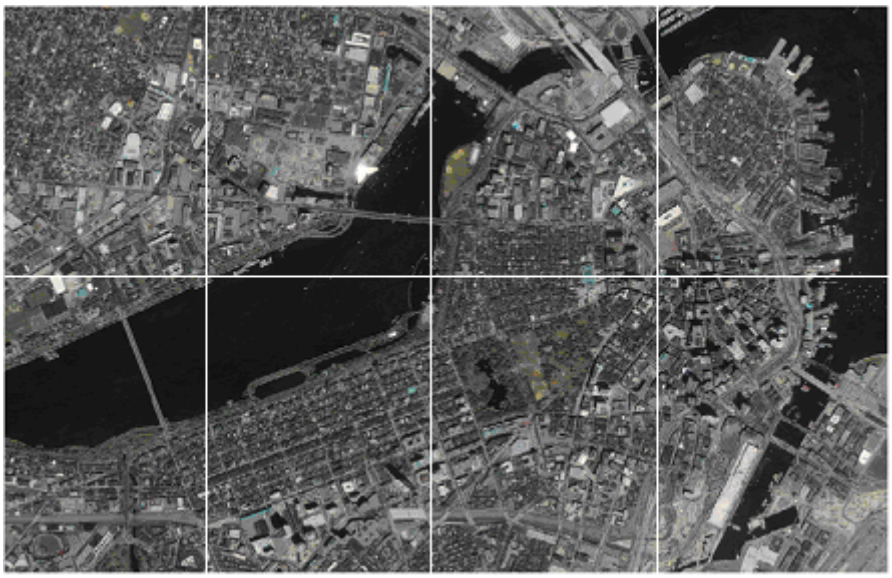

Figure 7
Partitioned digital photo of Boston

This computation required is 4.813 sec.

Now, computing the color approximation of each subimage simultaneously, in parallel way, using **ParallelSubmit** takes only 6.64 sec. (Figure 8)

DistributeDefinitions[**ColorQuantize**, pP]

pS = **WaitAll** [**Map** [**ParallelSubmit** [**ColorQuantize** [#,10]] & ,pP]]

Putting the parts together takes 0.11 sec,

pR = **ImageAssemble**[**Partition**[pS,4]]

Table 4
Performance of the computation of color quantization.

Comput. type	Time (sec.)	Speedup	Efficiency (%)	Relative unbalance
Non-parallel	23.19	-	12.5	-
Parallelize	6.64	7.01	43.66	0.26

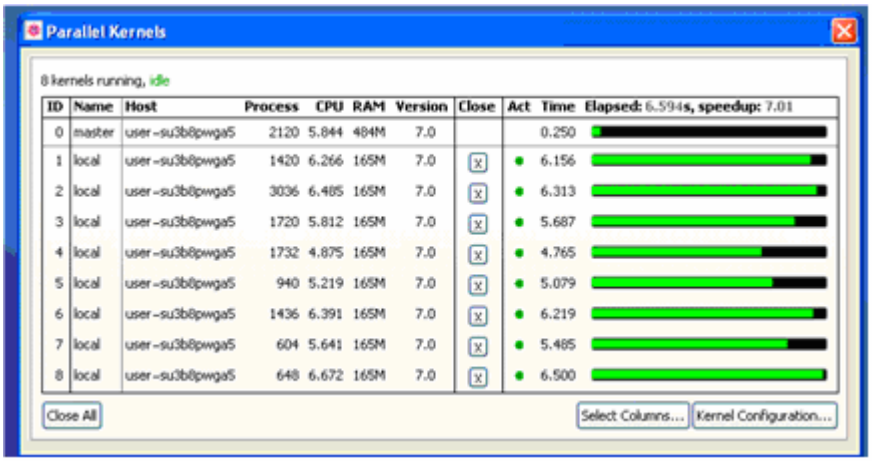

Figure 8
Status of the kernels in case of parallel evaluation of color quantization using **ParallelSubmit**.

The actual net win in running time becomes $\frac{23.19}{4.81+6.64+0.11} = 2.006$.

The performance metrics of this example are given in Table 4.

Conclusions

In this case study we have illustrated how *Mathematica* can manage time consuming algorithms parallel on a multicore machine. The considered three examples were analysed by the efficiency of the computation and the net reduction of the execution time.

Acknowledgement

This research has been supported by Hungarian National Scientific Research Foundation, Grant No. OTKA T69055.

References

[1] Intel Nehalem (microarchitecture): Performance and power improvements-Wikipedia, http://en.wikipedia.org/wiki/Intel_Nehalem_(microarchitecture)

[2] Haneberg W. C.: Computational Geosciences with Mathematica, Springer Berlin, 2004

[3] Awange J. L. and E. W. Grafarend: Solving Algebraic Computational Problems in Geodesy and Geoinformatics, Springer, Berlin, 2005

[4] Siddharth.S.:Parallel-computing-using-MATLAB,-http://sc08.sc-education.org/conference/engineering/mon/parallelmatlab2/ParallelMATLAB.pdf