jParalize – a simple, free and lightweight tool for parallelizing Matlab calculations on multicores and in clusters

Andrzej Karbowski[1],[2], Marek Majchrowski[1], and Piotr Trojanek[1]

[1] Institute of Control and Computation Engineering, Warsaw University of Technology, Poland, A.Karbowski@ia.pw.edu.pl
[2] NASK (Research and Academic Computer Network), ul. Wawozowa 18, 02-796 Warsaw, Poland

Abstract. We present a very simple, reliable, efficient and free tool for parallelizing calculations under Matlab in multicore and cluster environments. It does not use any compilers, MEX files, disk files, etc.. It is compatible with old Paralize package, but allows the involved cores/machines to do other jobs when a server is not busy.

Key words: Matlab, parallel software, free source software

1 Introduction

The aim of the described work was to add a free and lightweight support for distributed and parallel computations to the Matlab software. The main idea was to provide a simple user friendly tool (such as Matlab environment itself), which does not use too much the machines' resources to organize the calculations. Despite the release of Matlab Distributed Computing Toolbox/Engine several years ago there is still some sense in developing such tools, because:

- Matlab DCT/DCE are far from stability, that is there are many differences between the subsequent versions, also in the syntactics and semantics of the basic commands
- Matlab DCT itself, which is quite simple, allows for starting only up to 4 threads (workers/labs); to have more of them it is necessary to buy Matlab DCE, which is much more expensive and complicated in administration
- universities and research laboratories usually buy Matlab licenses in packs; this means, that after hours in labs many licenses are free.

Due to these reasons, we observe still a lot of interest in developing new software environments to parallelize Matlab, both on multicores and in clusters, to mention only Star-P among commercial, and Multicore - among free ones [2]. One of the oldest and - no doubts - the simplest software packages is Paralize [1]. It consists of only two m-functions: `paralize.m` and `serve.m` of, correspondingly, 146 and 94 lines (including comment lines). After the update made in 2006 by the first author, it helps to make fork-join calculations within a multiprocessor

or multicore machine as well in a cluster of computers. The biggest drawback of Paralize, as well as other free parallel packages like Multicore and MatlabMPI, is, that the communication and synchronization is realized by the disk files, what involves active polling, or, in other words, busy waiting.

The jParalize package is as simple as Paralize from the user point of view (actually it is compatible with it), but it does not waste the cycles of cores on active polling and allows to use the machines with started Matlab instances to other purposes.

2 Implementation

Distributing the work in *jParalize* is done by dividing data and executing the same operation on all chunks by slave Matlab processes (*solvers*). The data chunk together with function to be executed (*task*) is passed to *solvers* and results are gathered on the *console* Matlab session (Fig. 1).

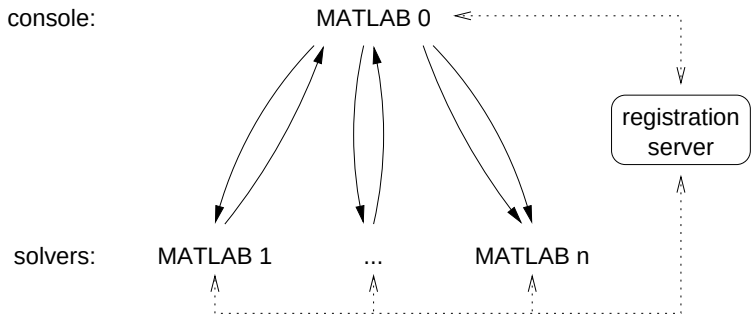

Fig. 1. *jParalize* communication model.

The presented approach has been implemented using Java language due to its portability and ease of interfacing with Matlab. Synchronization and data exchange between nodes is done by means of Java *RMI* (Remote Method Invocation).

The package consists of only three elements: *registration server*, *solvers*, *client*. The first process to be run is a *registration server*, whose task is to manage the set of free solvers. It is provided in a form of a single Java executable JAR file (which also contains all the Java classes to be used by Matlab). The next step is to run one or more *solvers* from within a Matlab session. It does not matter what operating systems are used, since both computing (Matlab) and communication (Java) environments are system independent. The registration is done by adding handle to a remote solver object to the managed set and then Matlab sessions are blocked until new tasks are available. Then the *client* just divides input data to chunks (as in *Paralize*, along the third dimension) and creates partial tasks to complete distributed job.

Solver is a blocking Matlab function, which waits on a Java object, until it is notified. Then, using provided methods, *solver* gathers all parameters, converts them from Java representation and makes calculation in Matlab. At the end results are converted and sent to the *jParalize client*.

In the cases, when the actual parameters are passed to some functions as Matlab structures, the package converts them to an internal representation, for example:

```
public class JMStruct
  implements Serializable {

  private Object fields, values;
  private int dimX, dimY;
  /* ... */
}
```

The variable *dimX* and *dimY* are also used to reshape the structure after Matlab→Java→Matlab conversion to match the original arrangement. In *fields* the package sustains the field names of the original Matlab structure and *values* are collected as in the original structure.

The complete implementation of `jParalize` has less than 500 lines of Java code and about 280 lines of Matlab files. It has been tested on Sparc Solaris and Intel compatible machines running Linux and Windows. It is important to note, that a job can be divided into solvers running on different operating systems and even different versions of Matlab and Java Virtual Machines.

3 Case study results

The tests have been performed on a network of Windows PCs with Intel Pentium 4 / 3GHz connected with 100Mbit/s network. All computers had common filesystem provided by Linux server running Samba service. The test job was to find a solution of a constrained optimization problem known as the Powell20 test problem [4]. It is defined as follows:

$$\min_x 0.5 \cdot (x_1^2 + x_2^2 + ... + x_n^2)$$

$$x_{k+1} - x_k \geq -0.5 + (-1)^k \cdot k; \quad k = 1, \ldots, n-1 \tag{1}$$

$$x_1 - x_n \geq n - 0.5$$

In tests the vector x was divided into p parts, what implied the corresponding division of the constraints; the p common ones were treated as global constraints and the Lagrange relaxation was applied to them. Then, the classical price method of the decomposed optimization was applied [3].

Actually, the only necessary work to parallelize the code was to replace the lines:

```
for i=1:p
   [xi(:,1,i),fi(1,1,i)]=pow20_pm_loct(xloc(:,1,i),lambda,ni,p,options_k);
end
```

with the line

```
[xi,fi]=paralize('pow20_pm_loct',xloc,lambda,ni,p,options_k);
```

The calculation times were as follows (n is the dimension of the problem, p - the number of processors):

Paralize

n \ p	2	4	6	8	12
108	17.7	12.7	23.9	NT	NT
216	39.5	33.1	40.4	NT	NT
432	366.8	100.6	89.9	109.1	NT
648	2033.2	299.7	192.6	**NA**	NT
864	NT	810.1	381.3	292.6	**NA**

jParalize

n \ p	2	4	6	8	12
108	16.1	15.8	21.7	NT	NT
216	40.4	31.5	41.0	NT	NT
432	331.6	99.5	84.5	98.2	NT
648	1832.7	287.2	183.5	546.9	NT
864	NT	786.8	377.7	282.6	288.0

In the above table NT means, that the calculations were not done for such p, n parameters; NA means, that we did not succeed to gather any result.

Comparing to the original *Paralize*, the proposed *jParalize* proved to be a bit faster, but more reliable and not paralyzing processors of the machines.

4 Conclusions

The main advantage of the presented free software is its: simplicity, reliability, ability to operate in heterogeneous network environments (especially where for administrative reasons common filesystem cannot be used). Avoiding active polling it does not waste the energy and does not block the cores waiting for tasks. In the case, when the transferred parameters and results are composed from complex data (e.g.: cells, mixed string–numeric structures, complex arrays), some additional time is necessary for conversion between Java and Matlab representations. However, this has only a minimal impact on the performance of the jobs, when they are split to a small number of heavy time consuming tasks.

References

1. Abrahamsson, T.: Paralize, adapted to Matlab v2006a+ by Andrzej Karbowski, `http://www.mathworks.com/matlabcentral/fileexchange/`
2. Choy, R. and A. Edelman: Parallel MATLAB: Doing it right. Proceedings of the IEEE. 93, 331-341 (2005)
3. Lasdon, L.S.: Optimization theory for large systems. Macmillan, New York (1970). republished by Dover, Mineola, N.Y. (2002).
4. Powell, M. J. D.: On the quadratic progamming algorithm of Goldfarb and Idnani. Mathematical Programmimg Study. 25, 46–61 (1985)