International Conference on Automation, Control, Engineering and Computer Science (ACECS'14)
Proceedings - Copyright IPCO-2014, pp.105-112
ISSN 2356-5608

Parallelization on heterogeneous platforms

Parallelization of polyhedron programs on heterogeneous platforms

Asma DAB[#1], Yosr SLAMA[#2]

[#] *Faculty of Sciences of Tunis, University of Tunis El Manar,*
University Campus - 2092 Manar II, Tunis, Tunisia
[1]asmadab1@yahoo.fr
[2]yosr.slama@fst.rnu.tn

Abstract— **The polytope model (PM) is a formalism which allowed systemizing and standardizing automatic parallelization of nested loops structured programs especially polyhedron programs (PP). It targeted, during its conception, the homogeneous (monolithic) parallel machines. The spread of numerous new parallel and distributed systems (e.g. clusters, grids, etc.) lead to the adaptation of certain concepts and approaches of the PM including techniques of scheduling, mapping and code generation, in order to take into account the specificities of these heterogeneous platforms. We establish in this paper a comparative study of several methods for PP parallelization, thus confirming the usefulness of adapting the MP for heterogeneous platforms. We also propose an approach to this adaptation, through two mapping methods, taking into account the characteristics of the target environment. An experimental study permits the validation of our contribution and the evaluation of its practical interest.**

Keywords— **Heterogeneous platform, Load Balancing, Mapping, Multicores, Polyhedron programs, Polytope model, Scheduling, Tiling.**

I. Introduction

In this paper, we investigate the parallelization of programs called polyhedron programs (PP) which are nested loops: Each loop's bounds are affine functions of enclosing loop indexes. Several approaches have been proposed in literature for the parallelization of such programs, some of which are placed in formal mathematics and are called polytope model (PM). Several approaches, based on the polytope model, have proven to be effective, but only when targeting machines with an infinite number of homogeneous processors. Today, machines are becoming more heterogeneous since the emergence of clusters and computational grids. Therefore, even machines which are originally homogeneous, such as multicores, are in reality heterogeneous when we consider the different "loads" that could have.

In this paper, in addition to a comparative study of different methods of parallelization on homogeneous and heterogeneous platforms known in literature, we propose new approach based on the PM and targets heterogeneous platforms. We begin by a state of the art of methods parallelizing polyhedron programs on homogeneous and heterogeneous platforms, then, we present the general principle of PM, after that, we introduce our parallelization approach for heterogeneous platforms which is detailed on two methods, and finally, we present an experimental study in order to validate our propositions on different heterogeneous platforms.

II. State of Art

We present here some methods of parallelization of loop nests targeting homogeneous and heterogeneous architectures. On homogeneous platforms, we quote, as static approaches, the binary search decision algorithm BSDA [1] and a technique for partitioning the iteration space called tiling [2,3,4,5]. This technique allows the partition of iteration space into tiles. Other methods of parallelizing loop nests have shown their effectiveness and fit within the PM such as the method of Feautrier and that of Darte and Vivien [6]. Among the dynamic approaches, we mention a class of methods called self-scheduling such as Chunk Self-Scheduling (CSS), Trapezoid Self-Scheduling (TSS) [7] and Guided Self-Scheduling (GSS) [8], Successive Dynamic Scheduling (SDS) [9], and Dynamic Scheduling Policy for SGRIDs [9]. On heterogeneous platforms, we find static methods as well as dynamic methods. As static approaches, we cite Adaptive Cyclic Scheduling ACS [8] suitable for homogeneous and heterogeneous systems, the Chain Pattern Scheduling CPS [10], and tiling of a sequence interchangeable loops [11]. Among the dynamic approaches, we quote the code optimization methods, such as the Multi-loop unrolling [12] which is improved later by introducing loop Fission and the Distributed Trapezoid Self-Scheduling DTSS [1]. In fact, DTSS is an extension of TSS on distributed platforms. The Dynamic Multi-Phase Scheduling for heterogeneous clusters DMPS [9] improves self-scheduling methods by adding synchronizations between the slaves. Based on the principle of the polytope model, the dynamic parallelization method proposed in the paper [13] allows the speculative parallelization of programs containing iterative structures but partially parallel.

III. General principle of the polytope model

The polytope model (PM) can be considered as a standard approach for automatic parallelization. Its mathematical foundation based on the concept of polyhedral allows it to include and standardize several techniques of classic loop nests parallelization [14]. The code generation in the polytope model assumes the availability of a sufficient number of processors (one processor for each instruction), named Pmax. When the number of available processors is less than Pmax, the iterations are distributed over the processors. The PM transforms the nest from a source program to a target parallel program through three steps:

Determination of the source polytope: The source polytope represents the iteration space of the source program. It needs to be written in the form of inequalities system satisfying the index loops ($Ax \le b$).

Determination of the target polytope: This step consists in segmenting the source polytope (A,b) in time intervals. This segmentation can be formulated as an affine transformation of the source polytope which preserves data dependencies and checks one or more performance criteria. Once determined, the transformation (from a scheduling and / or affine mapping) will be applied to the source polytope giving rise to the target polytope (AT^{-1},b). This affine transformation is represented by an invertible matrix T.

Determination of the target program: This last step generates the target program based on the target polytope. Two types of parallelism are distinguished: synchronous parallelism (time loops include space loops) and asynchronous parallelism (space loops include time loops).

Example 1:

Let consider the source program (P):

```
(P)DO x1=0, n
    DO x2=0, n
       A[x1,x2]= A[x1,x2] +
A[x1+1,x2-1]
     ENDDO
```

- The Source Polytope of (P): It is represented by the following inequalities system:

$$\begin{bmatrix} -1 & 0 \\ 0 & -1 \\ 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \le \begin{bmatrix} 0 \\ 0 \\ n \\ n \end{bmatrix}$$

- The scheduling which preserves dependencies and minimizes the execution time θ (x_1, x_2) = x_1 is represented by the matrix [1 0]. A valid allocation that represents the chosen scheduling and minimizes communications between the processors is Π (x_1, x_2) = $x_1 + x_2$. Thus, a space-time transformation can be represented by the square matrix: $T = \begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix}$, where the first line represents θ and the second one represents Π.

- T is unimodular, its inverse is the matrix: $T^{-1} = \begin{bmatrix} 1 & 0 \\ -1 & 1 \end{bmatrix}$. The target polytope is (AT^{-1},b) i.e

$$\begin{bmatrix} -1 & 0 \\ 1 & -1 \\ 1 & 0 \\ -1 & 1 \end{bmatrix} \begin{bmatrix} t \\ p \end{bmatrix} \le \begin{bmatrix} 0 \\ 0 \\ n \\ n \end{bmatrix}$$

Where t and p are the new loop counters respectively time and space (cf. Fig. 1).

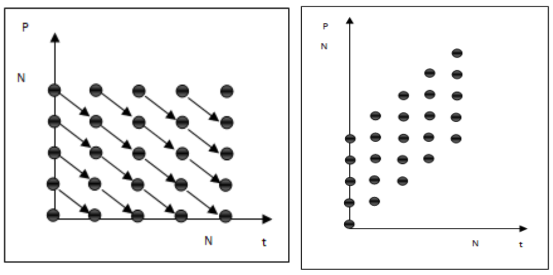

Fig. 1. Iteration space and dependencies of (P) before and after transformation

- Target Program: The program can be synchronous (1) or asynchronous (2):

```
(1)DOSER t=0,n
     DOPAR p = t, t+n
      A[t, p-t]=A[t, p-
     t]+A[t+1,p-t-1]
     ENDDOPAR
   ENDDOSER
```

```
(2)DOPAR p = 0, 2n
    DOSER t=max(0,p-n),min( n,
         p)
       A[t, p-t]=A[t, p-
     t]+A[t+1,p-t-1]
    ENDDOSER
  ENDDOPAR
```

IV. Parallelization Method for Heterogeneous Platform

The proposed approach is a static coarse grained approach derived for the extension of the polytope model to heterogeneous platforms. The aim of our approach is to balance distribution of loads among the processors to minimize the execution time (makespan). Given a target program resulting from the application of the polytope model, we apply a preprocessing to the generated code that will be used later as input to our approach. The approach takes also as input a description of the heterogeneous platform (speed and load of each processor). The output is a parallel code well adapted to the heterogeneity of the target platform. As the last stage of the PM parallelization allows generating two forms of target program, namely the synchronous and asynchronous program, we propose two methods each suites to one of these program forms: MHSP mapping heuristic for synchronous program and MHAP mapping heuristic for asynchronous program. We consider some assumptions which are:

(H1): All The processed program instructions have the same cost. Thus, we do not consider the case of tasks heterogeneity.

(H2): The target system is a heterogeneous system which consists of a fixed number p of processors (P_1, P_2, ..., P_p) having different speeds (V_1, V_2, ..., V_p) and different loads (number of processes running or waiting to be run on the processor) (Q_1, Q_2, ..., Q_p).

We calculate the available computing power of each processor P_i, A_i noted as follows:

$$A_i = \frac{V_i}{Q_i+1};$$

Here we add 1 to Q_i to avoid division by 0 Q_i=0). The total available computing power of the target system is denoted as follows:

$$A = \sum_{i=1}^{p} A_i$$

A. *Preprocessing*

The target program generated using polytope model for homogeneous environment can be an imperfect nest (instruction are not necessary enclosed in the inner loop) and can have non-affine bounds (min and max). Thus, we propose a preprocessing step transforming the program into a standard form in order to facilitate the application of our approach.

Stage 1: Linearization of non-affine bounds

This step transforms input programs containing min and max bounds to a nest which all bounds are affine using loop bursting.

Stage 2: Addition of missing space and time loops. This step ensures that any instruction is included by one or more space loops. It consists of adding, if necessary, loops whose lower bound equals the upper bound equal.

B. *Adaptation to heterogeneous platform*

Starting from a program having undergone pretreatment phase the goal is to adapt the program to the heterogeneity of the platform. According to the form of program (synchronous or asynchronous), we resort to a specific method of placement of instances iterations on processors. Here we present two methods MHSP (for synchronous program) and MHAP (for asynchronous program).

1) MHSP method

We denote N (t, i) the number of iterations assigned to processor P_i at time t and N (t,*) the total number of iterations at time t. For fixed t, the problem is formalized as an integer linear program whose variables are N (t, i), i = 1 .. p. The constraints are:

$$\begin{cases} \sum_{i=1}^{p} N(t,i) = N(t,*) \\ N(t,i) \geq 0, \forall 1 \leq i \leq p \end{cases}$$

The objectif function is to minimize

$$\max_{1\leq i\leq p} \frac{N(t,i)}{A_i}$$

The proposed strategy is to duplicate the parallel loop into two loops: a parallel loop for mapping on parallel processors and a sequential nested loop allowing each processor to compute its block of assigned iterations. We give here the forms of synchronous program before and after application of MHSP.

<u>Synchronous program before MHSP</u>

```
DOSER t = Lt , Ut
     DOPAR  proc = Lp,Up
        S(t,proc)
      ENDDOPAR
 ENDDOSER
```

<u>Synchronous program after MHSP</u>

```
DOSER t = Lt , Ut
  DOPAR  proc = 1, p
    DOSER index = start(proc) , end(proc)
          S(t,proc)
    ENDDOSER
  ENDDOPAR
ENDDOSER
```

The bounds of *index* are introduced to represent the start and the end of the block assigned to processor *proc*. Processors loads are calculated using the load partition algorithm (A).

```
Load partition algorithm (A)
----------------------------
Input: p processors, {Ai, i=1..p}, N(t,*)
Output: table containing processor loads N[ ]
Nrest=0
For each processor i do
        N[i] ← $\lfloor N(t,*) * \frac{Ai}{A} \rfloor$
end
sort available computing powers in a decreasing order
For each processor i=1:p do
    if Nrest = 0 then stop
    else
  a=0
        if i=1 then
          N[1]+= $\min\left(\left\lceil Nrest * \frac{A1}{A} \right\rceil , Nrest\right)$
          Nb=$\min\left(\left\lceil Nrest * \frac{A1}{A} \right\rceil , Nrest\right)$
        else
            for j= 1: i-1 do
                a=a+$\left\lceil Nrest * \frac{Ai}{A} \right\rceil$
            end
            N[i]+ = $\min\left(\left\lceil Nrest * \frac{Ai}{A} \right\rceil, Nrest - a\right)$
            Nb= $\min\left(\left\lceil Nrest * \frac{Ai}{A} \right\rceil, Nrest - a\right)$
         End if
         Nrest=Nrest – Nb
     End if
End
```

Example 2

The following example shows an iteration space before and after partitioning with MHSP method. The indexes start and end of blocks assigned to processors are grouped in Table 1. The platform used contains three processors of P1 (V1 = 1, Q1 = 0), P2 (V2 = 8, Q2 = 1) and P3 (V3 = 6, Q3 = 2) having total power A = 7 (A=A1+A2+A3=1+4+2=7).

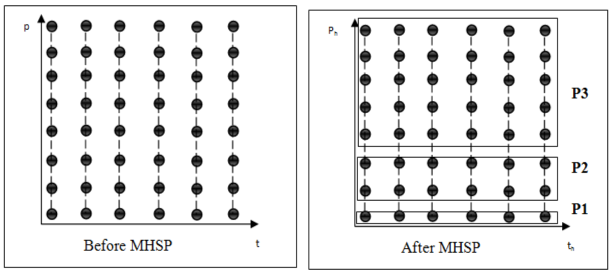

Fig. 2. MHSP effect on iteration space

TABLE I. START AND END INDEXES OF BLOCKS ASSIGNED TO PROCESSORS

Processor	Strat block index	End block index
P1	1	1
P2	4	8
P3	2	3

2) *MHAP Method*

The problem is formalized as an integer linear program whose variables are the N (i), i = 1 .. p. The constraints are:

$$\begin{cases} \sum_{i=1}^{p} N_i = P_{max} \\ N_i \geq 0, \forall\, 1 \leq i \leq p \end{cases}$$

The objectif function is to minimize

$$\max_{1 \leq i \leq p} \frac{Ni}{A_i}$$

The MHAP approach assigns to each processor a set of successive iterations, named block, whose size is proportional to the processor's computing power. Load balancing carried out at two levels: the first determines the assigned block size to each processor and the second adds an explanation of the choice of the block (taking into account data locality). When the difference between the time loop bounds is a constant function f, only the first level is applied. Otherwise, block choice depends on the sign of f.

The proposed strategy is to duplicate the parallel loop into two loops: a parallel processor mapping loop and a serial loop allowing each processor to compute its assigned iteration block. Below are the source code and the MHAP modified code.

Asynchronous program before MHAP

```
DOPAR  proc= 1 , p
  DOSER  t= Lt, Ut
```

Asynchronous program after MHAP

```
DOPAR  proc= 1 , p
 DOSER  index=start (proc) , end(proc)
    DOSER  t= Lt, Ut
     S(t,proc)
   ENDDOSER
 ENDDOSER
ENDDOPAR
```

International Conference on Automation, Control, Engineering and Computer Science (ACECS'14)
Proceedings - Copyright IPCO-2014, pp.105-112
ISSN 2356-5608

Let Pmax be the maximum number of processors used in a homogeneous virtual machine. The index bounds are the beginning and the end of the block assigned to processor proc which are calculated using the load partition algorithms (A) and (C).

```
Load partition algorithm (C)
-----------------------------------------------
Input: p processors, {Ai, i=1..p}, $P_{max}$, $Ut$, $Lt$
Output: table containing processor loads N[ ]
For each nested loop do
    $f(p)=Ut-Lt$
    if $f$ is a constant function then
        Call algorithm partition (A) replacing $N(t,*)$ with $P_{max}$
        Assign blocks to processors
    Else if
        if $f(p)$ is a positive function then
            Apply a descending sort processors
        Else if $f(p)$ is a negative function then
            Apply an increasing sort processors
        End if
        Call algorithm partition (A) replacing $N(t,*)$ with $P_{max}$
        Assign blocks to sorted processors
    End if
End for
```

Example 3

The following example shows an iteration space before and after partitioning approach MHAP. The indexes start and end blocks assigned to processors are grouped in the following table. The platform used consists of P1 (V1 = 1, Q1 = 0), P2 (V2 = 8, Q2 = 1) and P3 (V3 = 6, Q3 = 2) is of total power A =A1+A2+A3= 7.

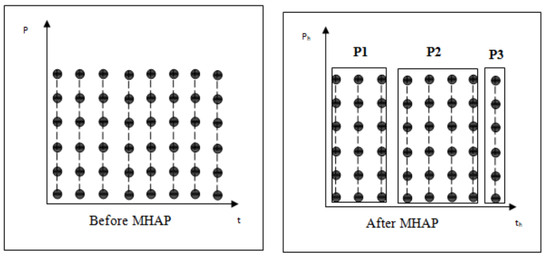

Fig. 3. MHAP effect on iteration space

TABLE II. START AND END INDEXES OF BLOCKS ASSIGNED TO PROCESSORS

Processor	Start block index	End block index
P1	1	3
P2	4	7
P3	8	8

V. EXPERIMENTAL STUDY

In this section, we present the target platforms and the technique used to make them heterogeneous. Then, we explain the experimental strategy adopted in our work. Finally, we describe and interpret the experimental results performed to investigate the performance of the proposed approaches by comparing them to the classical ones of parallelization using the polytope model and the GSS approach. The multi-core processors are still used as homogeneous environment since the cores have equal speeds. Given a multi-core machine, we will hide a portion (given as percentage) of the frequency of one or more cores to create an inter-core heterogeneity. The new cores frequencies will be denoted f_i ($1 < f_i < 100$) meaning that they operate at f_i%. The configuration will be re-presented by a n-uplet of frequencies, where n is the number of used cores. To test the proposed approaches, we used three types of multiprocessor machines: Dual Core Intel Core 2 Duo (denoted M2), a quad Core Intel i5CPU (denoted M4) and an Intel machine Xeon dual processor quad Core (M8). On each machine, we considered different configurations by changing each time cores mask. To measure the performance of the proposed parallel version V = {MHSP, MHAP} on a configuration C, we use metrics and ratio R:

$$\eta_V^C = \frac{\text{Parallel time of the homogeneous version on } C}{\text{Parallel time of the heterogeneous version } V \text{ on } C}$$

$$R(C) = \frac{\text{Parallel time of the version modified by MHAP on } C}{\text{Parallel time of the version modified by MHSP on } C}$$

To evaluate our approaches, we implement MHAP (A_HET), MHSP (S_HET), classical versions (synchronous (S_HOM) and asynchronous (A_HOM) parallel programs) and the GSS approach on numerous configurations of three platforms. The charge equilibrium proposed algorithms of are tested using matrix vector product MVP and Gauss Elimination. Here we present some results on four configurations (a homogeneous configuration (without using masks) and three configurations of three heterogeneous machines used).

A. *Validation of MHAP method*

To measure the performance of the MHAP version, we use the metric. The following figure represent some experiments of PMV parallel program modified with MHAP on configurations: (100,100) and (100,1) on platform M2, (100,50,1,100) on platform M4 and (100, 10, 10, 10, 20, 10, 10,30) on platform M8.

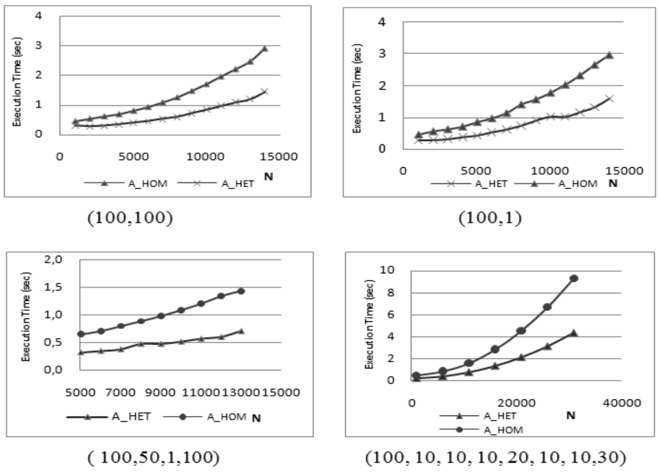

Fig. 4. . Execution time of A_HET and A_HOM on 4 configurations

The parallel program execution time of generated by the approach MHAP increases with the size N of the problem (size of the input matrix), a linear way, even sub-linear in most cases. The MHAP (A_HET) approach resulting program is faster than the classical approach generated by the classical approach targeting homogeneous platform (A_HOM). Even on a homogeneous configuration, the execution time A_HET is always below those of A_HOM. The MHAP approach adopts a different block size static distribution, which reduces the number of cache misses and thus improves the execution time. To calculate load balancing, we consider the configuration (100,50,20,30). The available computing powers are summarized in the following table.

TABLE III. AVAILABLE COMPUTING POWERS IN THE CONFIGURATION (100,50,20,30)

	P0	P1	P2	P3
Mask(%)	100	50	20	30
A (Go)	3.33	1.665	0.666	0.999
A (Mo)	3409.92	1704.96	681.984	1022.976

The ratio of computation time of processors P0 and P2 (respectively regarded as the slowest and fastest) is: (T [P2]) / (T [P0]) = 0.999. This calculated ratio is very close to 1 reflecting and thus being very close to the optimal one. This result is confirmed in all configurations, especially when the size of matrix N is large enough.

B. MHSP method validation

We present in this section some results of tests (MVP) on four configurations: (100,100,100,100), (75,50), (100,50,1,100) and (1,1,1,100,100,100,100,100). To measure MHSP version performance, we use therefore the metric

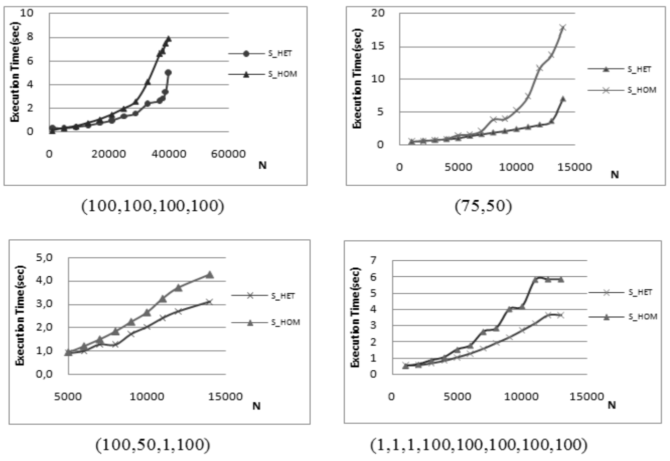

Fig. 5. Execution time of S_HET and S_HOM on 4 configurations

The program resulted from the approach MHSP (S_HET) is more efficient than that generated by the classical approach which targeting homogeneous platform (S_HOM). Even on a homogeneous configuration, the execution time of S_HET is always below those of S_HOM.

C. *Comparison of proposed approaches*

We propose here a comparison between the approaches MHSP and MHAP, by considering MVP. To compare our approach with an approach from literature, we choose a dynamic scheduling method belonging to the family self-scheduling which is Guided Self-Scheduling (GSS). We tested the three approaches on many configurations but we limit ourselves to present some representative results. To compare the approaches with the GSS version, we introduce two performance metrics denoted relative speed $\alpha_{r_{MHSP}}^{C}$ and $\alpha_{r_{MHSP}}^{C}$.

$$\alpha_{r_V}^{C} = \frac{\textit{Parallel time of the version V on C}}{\textit{Parallel time of the version GSS on C}}$$

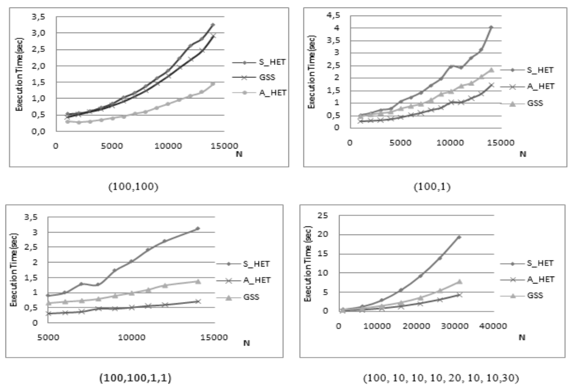

Fig. 6. Execution time of S_HET, A_HET and GSS on 4 configurations

We notice that even on a homogeneous configuration, MHAP approach is the best. This can be explained by the fact that our approaches affect consecutive blocks, thus reducing cache misses. We also notice that the ratio R in most configurations is about 4 (see table 4).

TABLE IV. EXECUTION TIME OF THREE APPROACHES FOR P=4 AND P=8

N	S_HET	A_ HET	GSS	$\alpha_{r_{MHSP}}^{t}$	$\alpha_{r_{HP}}^{t}$	R
			p=4 ; (100,100,1,1)			
5000	1.737	0.472	0.908	0.523	1.923	3.680
10000	0.906	0.316	0.665	0.734	2.105	2.867
14000	1.737	0.472	0.908	0.523	1.923	3.680
			p=8 ; (100,10,10,10,20,10,10,30)			
21000	9.234	2.156	3.637	0.394	1.687	4.283
26000	13.869	3.155	5.449	0.393	1.727	4.396
31000	19.371	4.393	7.766	0.401	1.768	4.410

For p = 8, the GSS method gives better results than the MHSP approach. This is clear through the relative acceleration which is in the order of 0.5. We also notice that MHAP vs GSS relative acceleration is about 2. In fact, this may be explained by the manner with which the two approaches calculate block sizes. The GSS approach affects the blocks in a dynamic way by reducing the size until reaching an iteration block, while the MHAP approach affects blocks once for all processors in a fair manner basing on a valid mapping without using overlap avoiding, and thus expectations processors. The MHAP approach, although static, gives better results than those given by the dynamic approach GSS. We have shown that static load balancing can be more efficient than dynamic distribution blocks. However, the GSS approach is more efficient than MHSP.

VI. Conclusion

Although a variety of effective methods of parallelization have been proposed in literature, we found no method targeting heterogeneous platforms based on the polytope model. So, we are interested in the parallelization of loop nests by the polytope model, adapted to heterogeneous environments. Depending on the form of the parallel program generated by the polytope model, we developed two approaches denoted MHAP and MHSP, treating respectively asynchronous and synchronous cases. The two approaches take as input the generated code after a preprocessing stage, the number of processors and their available computing power. We noticed that even on homogeneous configuration, the proposed approaches are the best. By comparing the proposed approaches and comparing each one to the GSS approach, we have noticed that the MHAP approach outperforms the GSS approach that is better than MHSP approach. The way used by MHAP to affect blocks (asynchronous) gives better results than the MHSP approach. As a perspective, we tend to experiment our method on real heterogeneous platforms such as clusters, grids, etc. A future prospect can be to integrate our approaches within automatic parallelizing for polyhedron programs targeting heterogeneous platforms.

References

[1] A.T. Chronopoulos, S. Penmatsa, J. Xu and S. Ali. Distributed loop-scheduling schemes for heterogeneous computer systems. Concurrency and Computation : Practice and Experience. 771-785. 2006.

[2] A. Buttari, J. Langou, J. kurzak. Parallel tiled QR factorization for multicore architecture, Concurrency and Computation Practice and experience in Wiley Interscience. 20:1573–1590. 2008.

[3] M. Athanasaki, E. Koukis and N. Koziris. Scheduling of Tiled Nested Loop onto a Cluster with a Fixed Number of SMP Nodes. In 12th Euromicro Conference on Parallel, Distributed and Network-Based Processing (PDP'04). A Coruna, Spain. 2004.

[4] P. Boulet, J. Dongarra, Y. Robert and F. Vivien. Static Tiling for Heterogeneous Computing Platforms. Parallel Computing. 25(5):547-568. 1999.

[5] A. Großlinger. Some Experiments on Tiling Loop Programs for Shared Memory Multicore Architectures. In Programming Models for Ubiquitous Parallelism in Dagstuhl Seminar Proceedings. Germany. 2008.

[6] C. Lengauer. Loop parallelization in the polytope model. In Proceedings of the International Conference on Concurrency Theory, LNCS 715. 398–416. Hildesheim. 1993.

[7] T. H. Tzen and L. M. Ni. Trapezoid self-scheduling: A practical scheduling scheme for parallel compilers. IEEE Transactions on Parallel and Distributed Systems. 4(1):87–98. 1993.

[8] F. M. Ciorba, T. Andronikos, and G. Papakonstantinou. Adaptive cyclic scheduling of nested loops. In 7th Hellenic European Research on Computer Mathematics and its Applications (HERCMA'05). 2005.

[9] F. M. Ciorba. Algorithms Design for the Parallelization of Nested Loops. Phd thesis. National Technical University of Athens, School of Electrical and Computer Engineering, Department of Informatics and Computer Technology. Athens. 2006.

[10] F. M. Ciorba, T. Andronikos, I. Drositis, G. Papakonstantinou, and P. Tsanakas. Reducing the communication cost via chain pattern scheduling. In 4th IEEE Conference on Network Computing and Applications (NCA'05).186–196. Cambridge, Massachusetts, USA. 2005.

[11] N. Ahmed, N. Mateev and K. Pingali. Tiling imperfectly-nested loop nests. In Proceeding Supercomputing '00 Proceedings of the 2000 ACM/IEEE conference on Supercomputing (CDROM) IEEE Computer Society. Washington, DC, USA. 2000.

[12] Y. M. Lam, J.G.F. Coutinho, W. Luk P.H.W. Leong. Optimising Multi-loop programs for heterogeneous computing systems. In Proc Southern Programmable Logic Conference (SPL). 129 – 134. Sao Carlos. 2009.

[13] B. Pradelle, P. Clauss, Vers la parallélisation dynamique dans le modèle polyédrique. ICPS/LSIIT – Strasbourg university. 2009.

[14] Y .Slama. *Etude de la parallélisation des nids de boucles par le modèle polyédrique*, Phd thesis, Faculty of sciences of Tunis. Tunisia. 1999.