Automatic Performance Monitoring on YMP Family, Application at IDRIS/CNRS

Etienne Gondet, IDRIS/CNRS Supercomputing Center for Scientific Research, Bat 506, BP 167, 91403 Orsay Cedex, FRANCE.

ABSTRACT: *How to register vectorial performance of all the user programs running on your YMP computer? We have developed a tool based on hpmflop that has been used on our C90 at CNRS/IDRIS (French National Supercomputing Center for Scientific Research). We will discuss the implementation, pros and cons, and added value that such a tool can bring. This paper will present C90 performances by means of graphs and statistics on users' programs.*

1 INTRODUCTION

IDRIS is the French Supercomputing Center for Scientific Research belonging to the CNRS. It is equipped with an IBM scalar cluster, two Cray C9X, a Mass Storage System (EMASS, CONVEX and FileServ software) and a T3E. This paper describes a home-made tool to help with vectorization built on CRI HPMFLOP which is able to record automatically all execution performances on C9X.

2 PRINCIPLE AND REALIZATION

2.1 HPM : Hardware Processor Monitor

Most CRAY users know HPM can give, after a run, a vectorial performance summary. This can be done without any overhead, essentially because it is built on hardware counters.

Each counter has to pick up the number of times a specific event such as vector floating multiply occurs. From these basic hardware counters, HPM builds up a page of vectorization information.

2.2 HPMFLOP

The purpose of CRI HPMFLOP is to register systematically these hardware counter values at the end of each execution.

To start the mechanism, the system manager has to configure your loader (SEGLDR) by adding in /lib/segdirs/def_ld and /lib/segdirs/def_seg the following directive:

```
hardref=_hpmdumpg, trbk
```

Below is the typical line of data dumped in a log for every execution collected by HPMFLOP : hpmg which is a marker, date and time at the end of execution, the execution's PID, the user's UID, the name of the executable and the N basic hardware counters.

```
hpmg 844683885 5090 28622 a.out
162307404017 20592934599 135584687250
37297986 274395161180 17556749138
505341756 185403602 ...
```

N=32 on C90, J90, T90 and only 8 on YMP.

2.3 Rebuilding HPM from the basic hardware counter values

We now have to process this log to make it understandable and valuable. In this paper, we consider the case where N=32 and we cover the basic counters from B0 to B31. In the HPM report , the counters are divided into 4 groups:

- Group 0 : main block, counters B0 to B7.
- Group 1 : instruction hold issue, counters B8 to B15.
- Group 2 : instruction types, counters B16 to B23.
- Group 3 : vector operation types, counters B24 to B31.

These 32 basic counters of HPM are detailed in Figure 4 which represents a C90 HPM output.

But the most valuable HPM indications have to be recalculated using the following formulae:

- User time = T0 = B0 * Clock period
- Vector Floating ops/sec = T1 = (B27 + B28 + B29) / T0
- Floating ops/sec = T2 = (T1 + B22/T0)
- VEC mem. reference/sec = T3 = (B30 + B31) / T0
- Avg conflict/ref = T4 = B5 / T3
- B/T mem ref/sec = T5 = (B4-B23)/T0 -T3

- All vector instruction = T6 = $\left(B1 - \sum_{i=16}^{23} Bi\right) / (T0)$

- Avl = Average length = T7 = $\left(\sum_{i=24}^{31} Bi\right) / (T6 \times T0)$

2.4 Our favorite metrics

We have decided to reduce the volume of information to one line for all collected executions with the following information:

- Date and time of end of execution.
- Login.
- Name of the executables
- User time (T0).
- Vector Megaflops (MFlops=T1).
- Megaflops (Mflops = T2).
- Millions of Instructions per second (Mips = B1/T0).
- Instruction Buffer Fetch per second (IBFs = B3/T0).
- Average Conflict by Reference (Avg Conf/ref = T4).
- Average vector length (Avl = T7).

Below is the output of our home made tool called *hpmi*. This example shows the differences between a pretty good and a very bad vectorized code.

```
atlas:>hpmi -u rlab002
Date    Heure    Login   Mfs Temps   Mips  IBFs VMfs  Conf/ref   Avl   Nom
------- -------  -------- ----- -------  -------  ------ ------ ---------  ------  ------
Feb08  12:22   rlab002    20    845  106.77   0.04    13     0.26    5.15    pri1
Feb08  16:26   rlab002    20    841  106.66   0.04    13     0.26    5.15    pri1
Feb17  16:16   rlab002   668    964   30.42   0.01   668     0.07   94.02    ari
```

The commands output contains multiple alarms:

- MFlops have to be as high as possible, whereas MIPS have to be low.
- Sometimes the Vector Megaflops are lower than the Mflops, which is a good indicator of a non fully vectorized code (use profview and compilation listing).
- High IBFs (over 1.0) can be due to *spaghetti* codes : codes with a lot of control transfer such as branch instructions or subroutine calls (use flowview for further investigation and to inline appropriate subroutines).
- The Conf/ref is relevant for Memory conflicts. It is something to look for, especially on C90 because even well vectorized applications on CRAY 2 or ymp can generate a lot of conflicts on a C90 (use perfview).
- The Avl is a very important vectorization indicator complementary to MFlops. A value much lower than 128 can be a clue that this code did not vectorize on a big enough loop.

3 MOTIVATIONS AND BENEFITS

3.1 A post mortem and automatic vector performance tool without time overhead

There are several motivations to implement such a tool. The main one is to be able to benchmark codes in real production conditions and to say for example, this code reaches 400 MFLOPS for every 300 executions of 10 000 seconds each whatever the data set is.

Most of the time, a code cannot be benchmarked on any possible type of dataset because this would be too time consuming and often it is benchmarked on testsets which are not completely realistic. With HPMFLOP, we can have a post-mortem, overhead free and automatic registration of all the user code performances.

3.2 User information about vectorization

At our site, it is particularly important to have such a tool because we have a great number of users, about 1600 logins working on 450 scientific projects and each user can have several codes. So the objective is to offer a command which summarizes for them all their execution performances. Thus, they are able to know quickly their performances on their production datasets.

Another benefit is to alert people who have unexpectedly broken their vectorization because they have modified their code by adding new functions, which is not uncommon, especially when the code is rewritten or reused by someone else.

4 CAUTIONS

4.1 variable HPM_MT

You can use the environment variable HPM_MT to set the time limit above which you decide to record the vector performance summary. By default, this lower limit is only 5 seconds, and consequently you have a lot of artefacts like system cron or daemons which are not useful. We have decided at our site to set the limit to 600 seconds which is the interactive limit. This has two consequences:

- We do not register all the interactive runs.
- We do not register the smallest runs in batch.

With this site limitation you can be sure to have a very reasonable log size.

4.2 HPMFLOP limitations

In fact, due to the characteristics of this CRAYTOOL, it does not collect the performance summary of executions if:

- CPU Time limit is exceeded.
- They are killed either by the user or the system team.
- A floating point exception or an operand range error occurs.
- The system crashes.
- The executable comes from another site, academic or commercial, which did not configure the linker to HPMFLOP works.

4.3 Special caution: value of MFLOPS

One can argue about the interest of this metric because it is well known that it is not the perfect indicator for the performance of an algorithm. A more efficient algorithm can make less MFLOPS because it reduces the number of multiplications and additions needed to do the same work. So a higher MFLOPS does not necessarily indicate a better algorithm, but only that it is better suited to this type of computer.

Moreover, some algorithms such as lattice-gas never use multiplication and addition functional units but the vector logical,shift pop and integer adds units (see in Figure 4: B24, B25, B26). Therefore MFLOPS is zero because it only picks up multiplications and additions. The following example describes the group counter 3 for a lattice-gas application :

B24 : Vector Logical	: 413.67M
B25 : Vector Shift/Pop/LZ	: 079.53M
B26 : Vector Integer Add	: 264.11M
B27 : Vector Floating Multiply	: 000.00M
B28 : Vector Floating Add	: 000.00M
B29 : Vector Floating Reciprocal	: 000.00M
B30 : Vector Memory Read	: 448.69M
B31 : Vector Memory Write	: 149.56M

It proves that a 0 MFlops code can be well vectorized anyway.

We recommend our users to always consider the metric MFLOPS with the Average Vector Length (Avl=T7,see 2.3) which considers all the vector operations detailed in the counter group 3 and not only the additions and multiplications.

Another solution would be to create a new metric resuming the mathematic vector operations better than MFlops called MVops (Million of Vector OPeration per second) which would be the sum of B24 to B29. In the above example the MFlops are 0 but the MVops would be 757.31 MVops.

4.4 What HPMFLOP cannot do

It can only study vectorization. It is a shame that such a tool does not record some very important information such as the following because it would be useful in a post mortem evaluation of an application:

- Maximum memory used.
- System time.
- Elapsed time.
- Multitask speedup.

This tool cannot check if a poor performance code hogs the memory. It is a big concern because we would like to focus on such a code first. This information is usually given by the Job Accounting (JA) and can be found later with the CRI CSAJREP tool.

5 STATISTICS

The percentage of recorded hours is around 66%. A lot of hours are not recorded mostly due to the value of the environment variable HPM_MT we have chosen (see 4.1) to avoid the lowest CPU time runs.

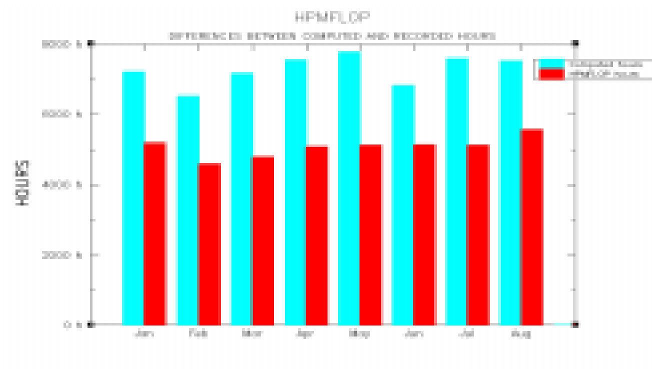


Figure 1: Comparison between hours computed and recorded by HPMFLOP during the year 1996.

The IDRIS C90 overall performance has progressed significantly to reach 300 MFlops this year. It is interesting to notice that the C90 average performance almost reaches the peak performance of its predecessor the CRAY-YMP (330 MFlops peak). Furthermore, Figure 2 shows seasonal performance variations probably due to the CPU time allocation made in June and December. This type of graph can help to study the performance evolution after a programming environment (cf77 to f90) or an operating system (Unicos 7 to Unicos 9) upgrade .

.

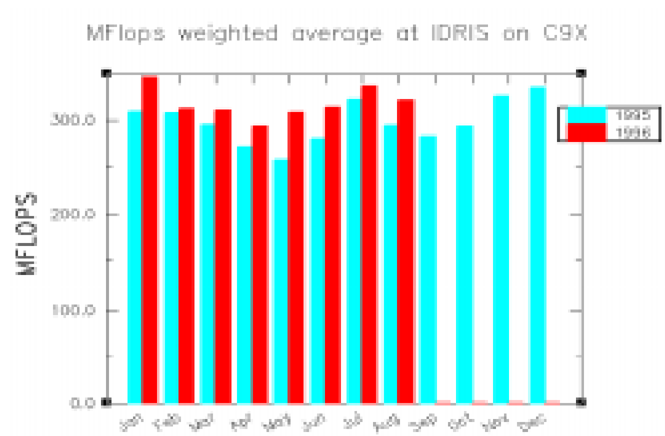


Figure 2: MFlops weighted average at IDRIS on C9X.

There are very few codes over 600 MFlops. The large number of codes between 400 and 600 Mflops is due to the use of the CRAY scientific library (LIBSCI). The significant number of hours under 100 Mflops is a result of some poorly vectorized codes but also to well vectorized codes for which MFlops is not the approriate metric. The variety of thematics dealted with on our C9X explains the diverse performances obtained

.

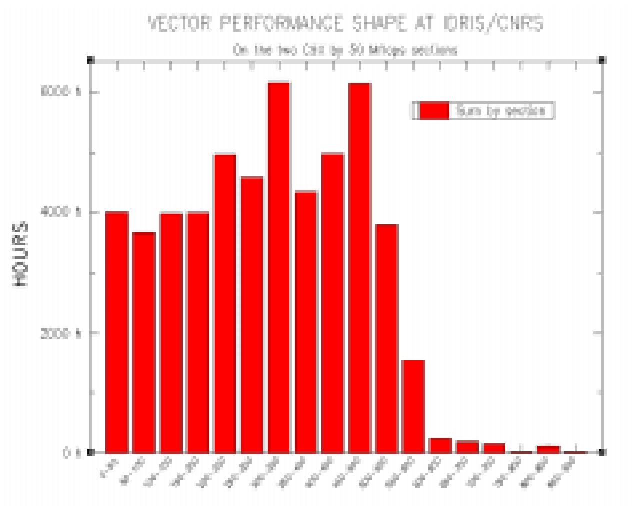


Figure 3: Performance shape at IDRIS/CNRS on C9X by 50 MFlops sections.

6 SUMMARY

The main advantage offered by HPMFLOP is the absence of overhead. This allows us to use it to dump systematically all user program performances thus eliminating the need to benchmark all codes.

The tool we have developed with the selected metrics detailed in this paper enables us to focus on the poorest codes and in quite a lot of cases, also provides valuable clues as to why an application is performing so poorly. Furthermore, with this tool, we have carried out a vectorial performance assessment for a lot of scientific projects over the whole year. This assessment is used by the scientists who are in charge of distributing the IDRIS/CNRS computing resources to determine the annual CPU time allocation of all the scientific projects.

As HPMFLOP was only installed in october 1994, it is quite difficult to assess precisely how much we have gained from it but we are already pleased to note an improvement. Our main expectation was not only to increase the C90 overall performance but also to gain on *human* efficiency by providing IDRIS and all users with a useful tool.

7 Acknowledgements

I would like to thank Nina Suvanphim and Philippe Tesson from Cray Research France, Claude Mercier and Jean-Marie Teuler from IDRIS for assisting me with this paper. Last but not least, my special thanks to Jean-Philippe Proux, Raphael Medeiros and Hervé Delouis from IDRIS, who helped me in developing this tool.

8 URL

http://www.idris.fr/docs/docu/publication/CUG_fall_96/hpmflop.html
http://www.idris.fr/docs/docu/publication/CUG_fall_96/hpmflop.ps

CPU seconds	:	36.347	CP executing	:	*8723484782*	*= B0*
Million inst/sec (MIPS)	:	51.52	Instructions	:	*1872762914*	*= B1*
Avg. clock periods/inst	:	4.66				
% CP holding issue	:	69.40	CP holding issue	:	*6054307103*	*= B2*
Inst.buffer fetches/sec	:	0.50M	Inst.buf. fetches	:	*18165134*	*= B3*
Floating ops/sec	:	357.09M	F.P. ops	:	**12979314669**	
Vector Floating ops/sec	:	356.32M	Vec F.P. ops	:	**12951435508**	
CPU mem. references/sec	:	294.21M	actual refs	:	*10694025583*	*= B4*
avg conflict/ref	:	0.11	actual conflicts	:	*1171758275*	*= B5*
VEC mem. references/sec	:	286.89M	actual refs	:	**10427906984**	
B/T mem. references/sec	:	3.44M	actual refs	:	**125112791**	
I/O mem. references/sec	:	0.47M	actual refs	:	*17034972*	*= B6*
avg conflict/ref	:	0.36	actual conflicts	:	*6084804*	*= B7*

Hold issue condition		% of all CPs	actual # of CPs	
Waiting on A-regs & access	:	7.26	*633593793*	*= B8*
Waiting on S-regs & access	:	3.43	*299081771*	*= B9*
Waiting on V-registers	:	27.72	*2418222991*	*= B10*
Waiting on B/T-registers	:	1.97	*172277900*	*= B11*
Waiting on Functional Units	:	43.59	*3802863252*	*= B12*
Waiting on Shared Registers	:	0.02	*1590265*	*= B13*
Waiting on Memory Ports	:	6.75	*588604460*	*= B14*
Waiting on Miscellaneous	:	3.01	*262362964*	*= B15*

(octal) instruction type		inst./CPUsec	actual inst.		% of all insts.
(000-004)Special	:	0.63M	*22766829*	*= B16*	1.22
(005-017)Branch	:	2.18M	*79344161*	*= B17*	4.24
(02x,030-033)A Register	:	29.12M	*1058293002*	*= B18*	56.51
(034-037)B/T Memory	:	0.21M	*7810320*	*= B19*	0.42
(040-043,071-077)S Register	:	1.70M	*61851944*	*= B20*	3.30
(044-061)Scalar Integer	:	1.65M	*59829294*	*= B21*	3.19
(062-070)Scalar Floating-Point	:	0.77M	*27879161l*	*= B22*	1.49
(10x-13x)Scalar Memory	:	3.88M	*141005808*	*= B23*	7.53
(140-177)All Vector	:	11.39M	**413982395**		22.11
			= MIPS - sum of groupe2		

type of vector operation		ops/CPUsec	actual ops	
Vector Logical	:	0.10M	*3600073*	*= B24*
Vector Shift/Pop/LZ	:	1.08M	*39329738*	*= B25*
Vector Integer Add	:	0.01M	*394786*	*= B25*
Vector Floaing Multiply	:	109.65M	*3985415816*	*= B27*
Vector Floaing Add	:	246.67M	*8966019659*	*= B28*
Vector Floaing Reciprocal	:	0.00M	*33*	*= B29*
Vector Memory Read	:	161.70M	*5877440650*	*= B30*
Vector Memory Write	:	125.19M	*4550466334*	*= B31*

Average Vecor Length for all Operations : **56.58** **= sum of groupe3 / All vector instruction**

Figure 4: HPM output with the 32 basics counters (B0 to B31). The underlined value have to be recalculated from them.

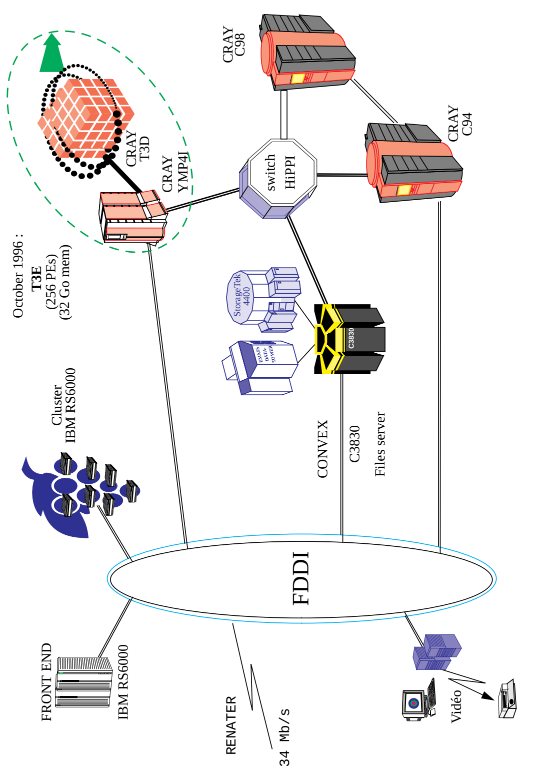
INSTITUT DU DEVELOPPEMENT
ET DES RESSOURCES
EN INFORMATIQUE SCIENTIFIQUE

HPMFLOP
PERFORMANCE MONITOR ON YMP AND C90 AT IDRIS

CNRS - IDRIS

Etienne GONDET: gondet@idris.fr

A) IDRIS/CNRS NETWORK



B) PRINCIPLE AND REALIZATION

- **Idea: to store automatically in a file at the end of an execution the HPM[1] record in order to find later any resumé of the vectorial performances of any execution.**

- **Characteristics: *historic*, automatic and *Post-mortem log* of vectorial performance execution.**

- **First operation on YMP:**

  ☞ You just have to replace:

  hardref=trbk by:
  hardref=_hpmdumpg,trbk

  ☞ in the system files:
  /lib/segdirs/def_ld and
  /lib/segdirs/def_seg

- **HPM counters are saved automatically if execution time < 600 seconds and if you link your object modules with the preceding options.**

1. Hardware Performance Monitor

C) WHAT WAS THE MOTIVATION?

- "**Benchmark**" **in real production conditions**

- **A lot of Cray users each year. About :**
  - ☞ **400 projects**
  - ☞ **1600 user accounts**

- **Each user has several programs and any program can have different types of input data.**
  - ☞ For example: g94 can do different tasks such as SCF, frequencies, CI, ...

- **Own-information-access for IDRIS users.**

- **Toolbox to help analyse various types of incidents for user support.**

D) HPMFLOP RECORD FORMAT:

- **ON C9X**

hpmg **date_time** **PID** UID **ecutable_name** ... + the **32** HPM basic counters (CPU Inst ...)

```
hpmg 785514293 96236 28355 emi2d2u.out 5998811552474
985605882535 4571073567462 5666241564 7681277450230
3253937907422 208605379 132153160 193288686358
389613224565 1651159610813 11334513276 1575882625950
9305084285 1858103170835 158118452521 35994311303
59145072703 407226847454 352914884 99294507402
123302358424 7406389956 25283325769 798091237929
217985995902 1367451907711 2903783524315 3828310832400
215349090299 4702064158717 2943951796293
```

  - ☞ From this basic information, you can reproduce the entire HPM from the CPU seconds to the last information: l'AVL[2].

- **ON YMP**

hpmg **date_time** **PID** UID **executable_name** ... + The **8** HPM basic counters (CPU Inst ...)

2. Average Vector Length

E) BASIC COUNTERS OF HPM ON A YMP

- **HPM on YMP only gives one of the 4 HPM blocks. That is the difference with the C90 which gives the 4 blocks at the same time:**
  - ☞ Group 0 : Main block.
  - ☞ Group 1 : Instruction hold issue.
  - ☞ Group 2 : Instruction types.
  - ☞ Group 3 : Vector operation types.

- **Each block has 8 basic counters.**

- **Main block:**

Group 0: CPU seconds	:	3.24	CP executing	:	540228141
Million inst/sec (MIPS)	:	50.28	Instructions	:	*162981187*
Avg. clock periods/inst	:	3.31			
% CP holding issue	:	54.90	CP holding issue	:	*296597851*
Inst.buffer fetches/sec	:	0.00M	Inst.buf. fetches	:	*2099*
Floating adds/sec	:	15.73M	F.P. adds	:	*50983161*
Floating multiplies/sec	:	15.69M	F.P. multiples	:	*50870999*
Floating reciprocal/sec	:	0.00M	F.P. reciprocals	:	*1*
I/O mem. references/sec	:	0.00M	I/O references	:	*0*
CPU mem. references/sec	:	47.26M	CPU references	:	*153179347*

Floating ops/CPU second : 31.42M

F) THE 32 BASIC COUNTERS OF HPM ON C9X

CPU seconds	:	36.347	CP executing	:	*8723484782*	=B0
Million inst/sec (MIPS)	:	51.52	Instructions	:	*1872762914*	=B1
Avg. clock periods/inst	:	4.66				
% CP holding issue	:	69.40	CP holding issue	:	*6054307103*	=B2
Inst.buffer fetches/sec	:	0.50M	Inst.buf. fetches	:	*18165134*	=B3
Floating ops/sec	:	357.09M	F.P. ops	:	12979314669	
Vector Floating ops/sec	:	356.32M	Vec F.P. ops	:	12951435508	
CPU mem. references/sec	:	294.21M	actual refs	:	*10694025583*	=B4
avg conflict/ref	:	0.11	actual conflicts	:	*1171758275*	=B5
VEC mem. references/sec	:	286.89M	actual refs	:	10427906984	
B/T mem. references/sec	:	3.44M	actual refs	:	125112791	
I/O mem. references/sec	:	0.47M	actual refs	:	*17034972*	=B6
avg conflict/ref	:	0.36	actual conflicts	:	*6084804*	=B7

Hold issue condition		% of all CPs	actual # of CPs	
Waiting on A-regs & access	:	7.26	*633593793*	= B8
Waiting on S-regs & access	:	3.43	*299081771*	= B9
Waiting on V-registers	:	27.72	*2418222991*	= B10
Waiting on B/T-registers	:	1.97	*172277900*	= B11
Waiting on Functional Units	:	43.59	*3802863252*	= B12
Waiting on Shared Registers	:	0.02	*1590265*	= B13
Waiting on Memory Ports	:	6.75	*588604460*	= B14
Waiting on Miscellaneous	:	3.01	*262362964*	= B15

(octal) instruction type		inst./CPUsec	actual inst.		% of all insts.
(000-004)Special	:	0.63M	*22766829*	= B16	1.22
(005-017)Branch	:	2.18M	*79344161*	= B17	4.24
(02x,030-033)A Register	:	29.12M	*1058293002*	= B18	56.51
(034-037)B/T Memory	:	0.21M	*7810320*	= B19	0.42
(040-043,071-077)S Register	:	1.70M	*61851944*	= B20	3.30
(044-061)Scalar Integer	:	1.65M	*59829294*	= B21	3.19
(062-070)Scalar Floating-Point	:	0.77M	*278791611*	= B22	1.49
(10x-13x)Scalar Memory	:	3.88M	*141005808*	= B23	7.53
(140-177)All Vector	:	11.39M	413982395		22.11
			= MIPS - sum of groupe2		

type of vector operation		ops/CPUsec	actual ops	
Vector Logical	:	0.10M	*3600073*	= B24
Vector Shift/Pop/LZ	:	1.08M	*39329738*	= B25
Vector Integer Add	:	0.01M	*394786*	= B25
Vector Floating Multiply	:	109.65M	*3985415816*	= B27
Vector Floating Add	:	246.67M	*8966019659*	= B28
VectorFloating Reciprocal	:	0.00M	*33*	= B29
Vector Memory Read	:	161.70M	*5877440650*	= B30
Vector Memory Write	:	125.19M	*4550466334*	= B31

Average Vecor Length for all Operations : 56.58 = sum of groupe3 / All vector instruction

G) FORMULA TO REBUILD A C90 HPM

- ❑ **Goal: to be able to complete the missing parts of HPM from an HPMFLOP record.**

- ❑ **Formulae for missing information on C90:**

☞ **Vector Floating ops/sec** = Vector Floating (Multiply + Add + Reciprocal)[3]

☞ Floating ops /sec = **Vector Floating ops /sec** + Scalar Floating_Point[4]

☞ **VEC mem. reference/sec** = Vector Memory (Write + Read)[5]

☞ avg conflict/ref = actual conflicts / **VEC mem. references/sec**

☞ B/T mem ref/sec=CPU mem ref/sec - VEC meme ref/sec - Scalar Mem[6]

☞ **All vector instruction** = MIPS - sum of counter group 2

☞ **Avl** = sum of counter group 3 / **All vector instruction**

3. In Counter group 3
4. In Counter group 2
5. In Counter group 3
6. In Counter group 2

CNRS - IDRIS
HPMFLOP: Carolina CUG *17 of October 1996*

H) METRIC IDEAS

- ❑ **Why not imagine your own (:-best-:) indicators:**

  ☞ **A** = Vector Floating **A**dd
  ☞ **M** = Vector Floating **M**ultiply
  ☞ **IBFS** = **I**nstruction **B**uffer **F**etch / **S**ec
  ☞ **MIPS** = **M**illion **I**nstruction **P**er **S**econds

- ❑ **What do the following indicators mean?**

☞ A / M

☞ (A - M) / (A + M)

☞ VEC memory references / FLOPS

☞ FLOPS / INST

☞ MIPS / IBFS

☞ FLOPS/IBFS

CNRS - IDRIS
HPMFLOP: Carolina CUG *17 of October 1996*

I) RESTRICTIONS ON HPMFLOP

PROS

- ❑ **HPM/HPMFLOP results are objective.**

- ❑ **HPMFLOP does not have any effect on execution time.**

- ❑ **HPMFLOP is automatic and easy to implement.**

CONS

- ❑ **What HPMFLOP/HPM cannot indicate :**

  ☞ Memory used.
  ☞ System time.
  ☞ Elapsed time.
  ☞ Multitasking speedup.

In fact the accounting information (ja).

- ❑ **These information can be found later by CSAJREP.**

CNRS - IDRIS
HPMFLOP: Carolina CUG *17 of October 1996*

J) REAL CASE: LATTICE-GAS

```
CPU seconds                 :    35.65    CP executing      :   [illegible]

Million inst/sec (MIPS)     :    30.25    Instructions      :   [illegible]
Avg. clock periods/inst     :     7.53
% CP holding issue          :    82.71    CP holding issue  :   [illegible]
Inst.buffer fetches/sec     :     0.01M   Inst.buf. fetches :   [illegible]
Floating ops/sec            :     0.00M   F.P. ops          :            1
Vector Floating ops/sec     :     0.00M   Vec F.P. ops      :            0
CPU mem. references/sec     :   590.36M   actual refs       :   [illegible]
    avg conflict/ref        :     0.38    actual conflicts  :   [illegible]
VEC mem. references/sec     :   590.35M   actual refs       :   [illegible]
B/T mem. references/sec     :     0.00M   actual refs       :            0
I/O mem. references/sec     :     0.15M   actual refs       :   [illegible]
    avg conflict/ref        :     0.89    actual conflicts  :   [illegible]

  Hold issue condition             % of all CPs     actual # of CPs
Waiting on A-regs & access     :   2.52              [illegible]
Waiting on S-regs & access     :   0.02              [illegible]
Waiting on V-registers         :   8.18              [illegible]
Waiting on B/T-registers       :   0.00                        5
Waiting on Functional Units    :  27.74              [illegible]
Waiting on Shared Registers    :   0.00              [illegible]
Waiting on Memory Ports        :  48.45              [illegible]
Waiting on Miscellaneous       :   2.01              [illegible]

 (octal) instruction type          inst./CPUsec    actual inst.  % of all insts.
(000-004)Special               :      1.03M       [illegible]     3.40
(005-017)Branch                :      1.03M       [illegible]     3.40
(02x,030-033)A Register        :     17.59M       [illegible]    58.06
(034-037)B/T Memory            :      0.00M                 0     0.00
(040-043,071-077)S Register    :      0.01M       [illegible]     0.03
(044-061)Scalar Integer        :      0.01M       [illegible]     0.02
(062-070)Scalar Floating-Point :      0.00M                 1     0.00
(10x-13x)Scalar Memory         :      0.02M       [illegible]     0.06
(140-177)All Vector            :     10.59M       [illegible]    35.02

  type of vector operation        ops/CPUsec      actual ops
Vector Logical                 :    413.67M       [illegible]
Vector Shift/Pop/LZ            :     78.53M       [illegible]
Vector Integer Add             :    264.13M       [illegible]
Vector Floating Multiply       :      0.00M                 0
Vector Floating Add            :      0.00M                 0
Vector Floating Reciprocal     :      0.00M                 0
Vector Memory Read             :    440.69M       [illegible]
Vector Memory Write            :    149.56M       [illegible]

Average Vector Length for all Operations : 127.95
```

CNRS - IDRIS
HPMFLOP: Carolina CUG *17 of October 1996*

K) CRON, SCRIPTS ET HPMI

- ❑ **Every night we process the raw log to make it understandable.**
- ❑ **For each months, we keep 2 logs:**
  - ☞ the raw log
  - ☞ the processed log.
- ❑ **HPMI: user command to display hpmflop information:**
  - ☞ RESTRICTIONS: a user can only see his own recorded execution performances when they are over 600 secondes.
  - ☞ USAGE: hpmi -u *login* [-g *groupe*] [-a 94]

Date	Heure	Login	Mfs	Temps	Mips	IBFs	VMflops	Conf/ref	Avl	Nom
-----	-----	-------	------	------	------	------	-------	----------	------	--------
Feb08	12:22	rlab002	20	845	106.77	0.04	13	0.26	5.15	pri1
Feb08	16:26	rlab002	20	841	106.66	0.04	13	0.26	5.15	pri1
Feb17	16:16	rlab002	668	964	30.42	0.01	668	0.07	94.02	ari

CNRS - IDRIS
HPMFLOP: Carolina CUG *17 of October 1996*

L) HPMI : Most important values in HPM

CPU seconds	:	36.347	CP executing	:	8723484782
Million inst/sec (MIPS)	:	51.52	Instructions	:	1872762914
Avg. clock periods/inst	:	4.66			
% CP holding issue	:	69.40	CP holding issue	:	6054307103
Inst.buffer fetches/sec	:	0.50M	Inst.buf. fetches	:	18165134
Floating ops/sec	:	357.09M	F.P. ops	:	12979314669
Vector Floating ops/sec	:	356.32M	Vec F.P. ops	:	12951435508
CPU mem. references/sec	:	294.21M	actual refs	:	10694025583
avg conflict/ref	:	0.11	actual conflicts	:	1171758275
VEC mem. references/sec	:	286.89M	actual refs	:	10427906984
B/T mem. references/sec	:	3.44M	actual refs	:	125112791
I/O mem. references/sec	:	0.47M	actual refs	:	17034972
avg conflict/ref	:	0.36	actual conflicts	:	6084804

Hold issue condition		% of all CPs	actual # of CPs
Waiting on A-regs & access	:	7.26	633593793
Waiting on S-regs & access	:	3.43	299081771
Waiting on V-registers	:	27.72	2418222991
Waiting on B/T-registers	:	1.97	172277900
Waiting on Functional Units	:	43.59	3802863252
Waiting on Shared Registers	:	0.02	1590265
Waiting on Memory Ports	:	6.75	588604460
Waiting on Miscellaneous	:	3.01	262362964

(octal) instruction type		inst./CPUsec	actual inst.	% of all insts.
(000-004)Special	:	0.63M	22766829	1.22
(005-017)Branch	:	2.18M	79344161	4.24
(02x,030-033)A Register	:	29.12M	1058293002	56.51
(034-037)B/T Memory	:	0.21M	7810320	0.42
(040-043,071-077)S Register	:	1.70M	61851944	3.30
(044-061)Scalar Integer	:	1.65M	59829294	3.19
(062-070)Scalar Floating-Point	:	0.77M	278791611	1.49
(10x-13x)Scalar Memory	:	3.88M	141005808	7.53
(140-177)All Vector	:	11.39M	413982395	22.11
			= MIPS - sum of group3	

type of vector operation		ops/CPUsec	actual ops
Vector Logical	:	0.10M	3600073
Vector Shift/Pop/LZ	:	1.08M	39329738
Vector Integer Add	:	0.01M	394786
Vector Floating Multiply	:	109.65M	3985415816
Vector Floating Add	:	246.67M	8966019659
VectorFloating Reciprocal	:	0.00M	33
Vector Memory Read	:	161.70M	5877440650
Vector Memory Write	:	125.19M	4550466334

Average Vector Length for all Operations : 56.58 = sum of group4 / All vector instruction

CNRS - IDRIS
HPMFLOP: Carolina CUG *17 of October 1996*

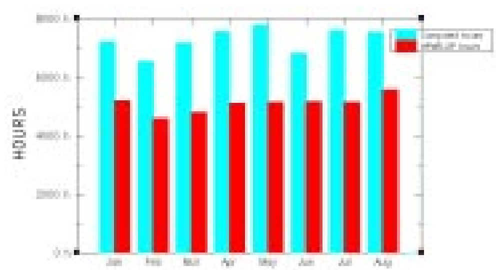
M) EMAIL RESUME AND ANNUAL ASSESSMENT

- ❑ **Each user account receive every 3 months, the result of hpmi by electronic mail.**
- ❑ **Each year, before thematic program comitee meetings, we produce an assesment form per project.**
- ❑ **We also resume MASS STORAGE occupied and BONUS/MALUS relative to memory used and speedup obtained.**

CNRS - IDRIS
HPMFLOP: Carolina CUG *17 of October 1996*

N) GRAPHS

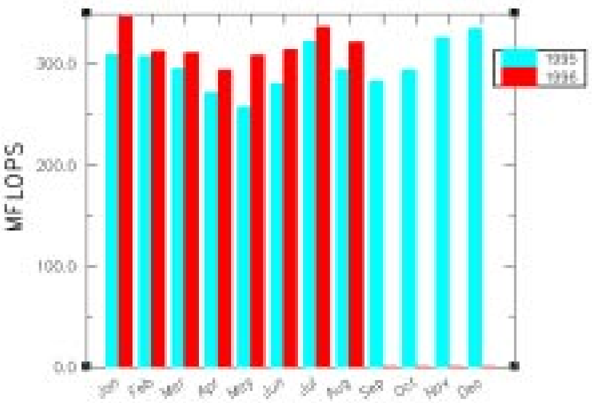
Comparison between hours computed and recorded by HPMFLOP during the year 1996.



- ❑ **The percentage of recorded hours is around 66%.**
- ❑ **A lot of hours are not recorded mostly due to the value of HPM_MT we have chosen.**

CNRS - IDRIS
HPMFLOP: Carolina CUG *17 of October 1996*

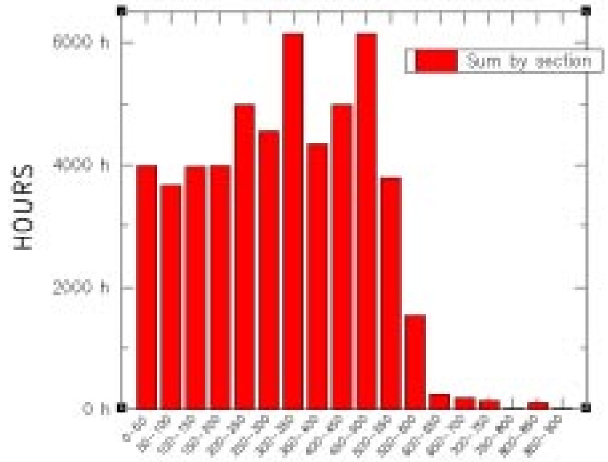
MFlops weighted average at IDRIS on C90



❑ The overall performance has progressed significantly to reach 300 MFlops this year.

CNRS - IDRIS
HPMFLOP: Carolina CUG 17 of October 1996

Performance shape at IDRIS/CNRS on C9X by 50 MFlops sections.



❑ There are very few codes over 600 MFlops. The large number of codes between 400 and 600 Mflops is due to the use of the CRAY scientific library (LIBSCI).

❑ The significant number of hours under 100 Mflops are due to some poorly vectorized codes but also to good vectorized codes for which MFlops is not the approriate metric.

CNRS - IDRIS
HPMFLOP: Carolina CUG 17 of October 1996