Roofline and Matrix Multiplication PAPI Analysis

José Alves, Rui Brito

Universidade do Minho

Braga, November 2012

Index

Sources for Machine Specifications

Sources used for a complete profile:

Linux System Information (/proc/cpuinfo, /proc/meminfo) To gather specifications on hardware;

Web (ark.intel.com, crucial.com) For micro architecture and memory specifications;

Linux Tools and Packages (dmidecode, sysctl, bandwidth) To gather memory, cpu and bandwidth info;

Machines Specs

Manufacter:	Apple
Model:	MacBook Pro late 2008
Processor	
Manufacturer:	Intel
Arch:	Core
Model:	Core 2 Duo T9600
Cores:	2
Clock Frequency:	2.80 GHz
FP Performance's Peak:	44.8 GFlops/s

Table: MacBook Pro late 2008 specifications

Machines Specs

Cache	
Level:	1
Size:	32KB + 32KB
Line Size:	64 B
Associative:	8-way
Memory Access Bandwidth:	40 GB/s
Level:	2
Size:	6 MB
Line Size:	64 B
Associative:	24-way
RAM	
Type:	SDRAM DDR3 PC3-8500
Frequency:	1067 MHz
Size:	4 GB
Num. Channels:	2
Latency:	13.13 ns

Table: MacBook Pro late 2008 specifications

Machines Specs

Manufacter:	HP
Model:	Pavillion dv6-2190ep
Processor	
Manufacturer:	Intel
Arch:	Nehalem
Model:	i7-720QM
Cores:	4
Clock Frequency:	1.60 GHz
FP Performance's Peak:	51.2 GFlops/s

Table: HP Pavillion dv6-2190ep specifications

Machines Specs

Cache	
Level:	1
Size:	32KB + 32KB
Line Size:	64 B
Associative:	4/8-way
Memory Access Bandwidth:	22 GB/s
Level:	2
Size:	256 KB
Line Size:	64 B
Associative:	8-way
Level:	3
Size:	6 MB
Line Size:	64 B
Associative:	12-way
RAM	
Type:	SDRAM DDR3 PC3-10600
Frequency:	1333 MHz

Problem

Analyse the performance of a **matrix multiplication** algorithm,

$$MatrixA * MatrixB = MatrixC \tag{1}$$

wich contains a triple nested loop with the indexes i,j and k (line,column and position).
The implementation used runs two versions of the problem, one multipying matrixA with matrixB, and another multipying matrixA with the transpose of matrixB.

Algorithm

Standard implementation of a matrix multiplication in C.

```
for (i = 0; i < size; i++) {
    for (j = 0; j < size; j++) {
        for(k = 0; k < size; k++) {
            acc += matrixA[i][k] * matrixB[k][j];
            }
            matrixC[i][j] = acc;
            acc = 0;
        }
    }
```

Counters Used

Used counters gathered by PAPI:

PAPI_TOT_CYC Total cycles;

PAPI_TOT_INS Total instructions

PAPI_LD_INS Load Instructions

PAPI_SR_INS Store Instructions

PAPI_FML_INS Multiply instructions

PAPI_FDV_INS Division instructions

PAPI_VEC_INS Vector Instructions

PAPI_FP_OPS Floating point operations

PAPI_L1_DCA L1 data cache accesses

PAPI_L1_DCM L1 data cache misses

PAPI_L2_DCA L2 data cache accesses

PAPI_L2_DCM L2 data cache misses

Test cases

Test cases were selected to fit on the multiple memory levels.
Each Test case was run 4 times for each version of the problem.

Memory	Size	Matrix Size
L1	30 KB	50
L2	255 KB	146
L3	3 MB	500
RAM	7.68 MB	800

Table: Test cases

Memory Accesses: Estimated Value

Code structure (based on Assembly analysis):

```
for(1 to N) {
    [4 stores, 1 load, 7 instr.]
    for(1 to N) {
        [6 stores, 3 loads, 34 instr.]
    }
    [3 stores, 6 loads, 26 instr.]
}
for(1 to N) {
    [3 stores, 3 loads, 10 instr.]
}
```

Memory Accesses: Formula

Based on the previous code structure, the number of memory accesses can be estimated with:

$$9N^2 + 20N$$

where N is the number of objects being processed.
The total estimated number of instruction is given by:

$$34N^2 + 43N$$

Memory Accesses

The following table shows the number of memory accesses, by PAPI readings and by estimation from the previous formula

Test	PAPI	Estimated	Est. Error	Accesses/Inst
L1_1	38716	38144	1.50%	0.27
L1_2	150588	150016	0.38%	0.27
L2_1	604144268	604143616	0.00%	0.26
L2_2	2416247656	2416246784	0.00%	0.26
RAM_1	38656024690	38656016384	0.00%	0.26
RAM_2	154621476174	154621444096	0.00%	0.26

Mult/Add balance

There is no counter for Add operations, so it was estimated with

$$PAPI\_FP\_INS - PAPI\_FML\_INS - PAPI\_FDV\_IN$$

Test	FP Mul Inst	FP Add Inst	Mul/Add balance
L1_1	33555	30395	90.58%
L1_2	133168	117874	88.52%
L2_1	537008683	469949801	87.51%
L2_2	2147782971	1879409199	87.50%
RAM_1	34360620752	30066293472	87.50%
RAM_2	137440660580	120261583165	87.50%

CPI

Calculated based on FPI_TOT_INS and FPI_TOT_CYC

Test	Instructions	Cycles	CPI	IPC
L1_1	143317	135750	0.95	1.06
L1_2	563861	472027	0.84	1.19
L2_1	2282055015	1686732377	0.74	1.35
L2_2	9127511619	6775496328	0.74	1.35
RAM_1	146031715237	171062387955	1.17	0.85
RAM_2	584121221418	687083107943	1.18	0.85

Miss rates

Based on the counters that give total accesses and misses to both L1 and L2 cache levels

Test	L1 Accesses	L1 Miss %	L2 Accesses	L2 Miss %
L1_1	50287	0.20%	290	33.79%
L1_2	194416	0.10%	476	34.03%
L2_1	750943668	11.18%	219910694	0.01%
L2_2	3007631681	11.18%	882123490	0.01%
RAM_1	48304577585	11.14%	10917610760	46.85%
RAM_2	194497858380	11.06%	42817380648	50.25%

Operational Intensity

Attempted to estimate Bytes read from RAM with L2_MISSES * 64, assuming that every miss will issue a new cache line read from RAM (64 Bytes) However:

- L2 counters are derived (maybe not reliable?)
- #L1 misses <> #L2 accesses. So we can also assume that #L2 misses <> #RAM accesses, making the previous formula wrong

Test	L2 Misses	Bytes from RAM	FP Inst.	Op. Intensity
L1_1	98	6272	68110	10.86
L1_2	162	10368	267551	25.81
L2_1	11434	731776	1074075321	1,467.77
L2_2	103001	6592064	4295643580	651.64
RAM_1	5114423222	0.3e+11 (0.3 TB)	68721939721	0.21
RAM_2	21517065599	1.3e+12 (1.3 TB)	274882221251	0.20

Actually, 1.3 TB is near the calculated value that should be read from RAM if there was no cache in between.

Conclusion

- Some difficulties measuring memory ceilings. The Roofline paper used a custom Stream benchmark, and provided no theorethical way to estimate those values;
- PAPI counters may sometimes differ from what is expected, especially when measuring memory traffic;
- TLP was not explored, and it would certainly prove beneficial and easili implemented for this particular algorithm;

Roofline and Matrix Multiplication PAPI Analysis

José Alves, Rui Brito

Universidade do Minho

Braga, November 2012