

Proyecto Intermedio : Midiendo MFLOPS en problemas matriciales

William Oquendo

Contents

1	Resumen	1
2	Instrucciones preliminares : Librerías y definición de los problemas	1
2.1	Instalación y prueba de PAPI	1
2.2	Operación A : Transpuesta de una matriz.	3
2.3	Operación B : Multiplicación de matrices.	3
2.4	Soluciones a comparar: cálculo directo, con librería, y cálculo por blocking, con y sin optimización	3
3	Estudio computacional	4
3.1	Performance de la técnica de blocking en función del tamaño del bloque	4
3.2	Performance en función del tamaño de la matriz, tanto para blocking como para eigen	4
4	Entregables	5
4.1	Informe	5
4.2	Repositorio	5
5	Ejemplo básico	6

1 Resumen

En este proyecto usted medirá el desempeño (*performance*) de algunas técnicas de optimización, comparadas con las originales, usando MFLOPS (Millones de operaciones de punto flotante por segundo). Esta es una medida que permite comparar de forma útil dos algoritmos en la misma máquina, pero que depende de la máquina. Por esta razón usted debe hacerlo siempre en el mismo computador. En principio, usted puede estimar el número de operaciones de punto flotante que ejecuta su algoritmo, pero en este caso vamos a usar la librería PAPI (Performance API) que le permite medir esto y muchas cosas mas de forma portable. Por esta razón deberá instalarla y verificar que funcione en la máquina en la que va a realizar sus pruebas. El objetivo final es que estudie cuántos MFLOPS obtiene en función del tamaño del bloque y del tamaño de la matriz. Esto le permitirá observar los límites de cache y de esta manera podrá comparar con las especificaciones técnicas de su procesador y ver si está obteniendo un desempeño apropiado o no de su algoritmo.

Deadline : [2019-11-08 Fri 11:00]

2 Instrucciones preliminares : Librerías y definición de los problemas

2.1 Instalación y prueba de PAPI

Se usará spack para instalar la librería. La versión de PAPI que usaremos será 5.7.0 , que es la última que se encuentra en spack.

Para instalarla debe ejecutar el comando

```
spack install papi@5.7.0 # aceptar cualquier mensaje que salga
```

Para activarla simplemente debe ejecutar el siguiente comando

```
spack load papi
```

Luego de haberla activado puede ejecutar las utilidades

```
papi_component_avail
papi_mem_info
```

Si tiene algún error en estos comandos, debe corregirlo antes de seguir.

Para hacer una prueba de su máquina y de la instalación, compile el siguiente código que está basado en un ejemplo de PAPI, `papi-5.7.0/src/examples/PAPI_flops.c` , y que le permite medir los MFLOPS de una función dada (en este caso se llama `your_code`)

```

/*****
 * This example demonstrates the usage of the high level function PAPI_flops *
 * which measures the number of floating point operations executed and the *
 * MegaFlop rate(defined as the number of floating point operations per *
 * microsecond). To use PAPI_flops you need to have floating point operations *
 * event supported by the platform. *
 *****/

/*****
 * The first call to PAPI_flops initializes the PAPI library, set up the *
 * counters to monitor PAPI_FP_OPS and PAPI_TOT_CYC events, and start the *
 * counters. Subsequent calls will read the counters and return total real *
 * time, total process time, total floating point operations, and the *
 * Mflops/s rate since the last call to PAPI_flops. *
 *****/

#include <stdio.h>
#include <stdlib.h>
#include "papi.h"

int your_code();

int main()
{
    float real_time, proc_time,mflops;
    long long flpops;
    float ireal_time, iproc_time, imflops;
    long long iflpops;
    int retval;

    /*****
     * if PAPI_FP_OPS is a derived event in your platform, then your *
     * platform must have at least three counters to support PAPI_flops, *
     * because PAPI needs one counter to cycles. So in UltraSparcIII, even *
     * the platform supports PAPI_FP_OPS, but UltraSparcIII only has two *
     * available hardware counters and PAPI_FP_OPS is a derived event in *
     * this platform, so PAPI_flops returns an error. *
     *****/
    if((retval=PAPI_flops(&ireal_time,&iproc_time,&iflpops,&imflops)) < PAPI_OK)
    {
        printf("Could not initialise PAPI_flops \n");
        printf("Your platform may not support floating point operation event.\n");
        printf("retval: %d\n", retval);
        exit(1);
    }

    your_code();

    if((retval=PAPI_flops( &real_time, &proc_time, &flpops, &mflops))<PAPI_OK)
    {
        printf("retval: %d\n", retval);
        exit(1);
    }

    printf("Real_time: %f Proc_time: %f Total flpops: %lld MFLOPS: %f\n",
        real_time, proc_time,flpops,mflops);

```

```

    exit(0);
}

int your_code()
{
    int i;
    double tmp=1.1;

    for(i=1; i<2000; i++)
    {
        tmp=(tmp+100)/i;
    }
    return 0;
}

```

Debe compilarlo sin optimización como

```
gcc -O0 -g papi_test.c -lpapi
```

Y después de ejecutarlo obtendrá algo como

```
Real_time: 0.000032 Proc_time: 0.000030 Total flops: 6021 MFLOPS: 200.699997
```

La idea es que el código anterior será la que usted use para hacer sus cálculos de medición de performance. Tenga en cuenta que al correr varias veces el ejecutable obtendrá valores fluctuantes, así que al final deberá tener una pequeña estadística para sacar el máximo MFLOP y ojalá correr sus procesos en una máquina no tan saturada.

Usted puede encontrar mas tutoriales buscando en google, con resultados como <http://www.drdobbs.com/tools/performance-monitoring-with-papi/184406109>.

Se le recomienda buscar el peak performance en MFLOPS o GFLOPS (Giga flops) del procesador que está usando, para normalizar sus resultados con este valor peak esperado.

2.2 Operación A : Transpuesta de una matriz.

Si usted tiene una matriz cuadrada $N \times N$ definida por sus elementos como $\mathbf{A} = A_{ij}$, su transpuesta se define como $\mathbf{A}^T = (A^T)_{ij} = A_{ji}$.

2.3 Operación B : Multiplicación de matrices.

Dadas dos matrices cuadradas A_{ij} y B_{ij} de tamaño $N \times N$, se define a la matriz C como la multiplicación entre A y B de la forma

$$C_{ij} = \sum_k A_{ik} B_{kj}. \quad (1)$$

2.4 Soluciones a comparar: cálculo directo, con librería, y cálculo por blocking, con y sin optimización

Usted deberá implementar de tres formas diferentes cada una de las operaciones anteriormente definidas, a saber:

Cálculo directo Se refiere a implementar la operación usando bucles sencillos, sin ninguna técnica extra. Por ejemplo, para implementar la transpuesta, el siguiente código sería suficiente (se asumen dos arreglos bidimensionales):

```

1      for (ii = 0; ii < N; ++ii) {
2          for (jj = 0; jj < N; ++jj) {
3              AT[ii][jj] = A[jj][ii];
4          }
5      }

```

Cálculo por blocking Se refiere al uso de la técnica de blocking (división de la matriz en pequeños bloques) para calcular la transpuesta o la multiplicación de matrices. Se espera que esta técnica incremente el performance. Debe investigar la técnica. En stack overflow hay varios ejemplos. Tenga en cuenta que cuando el tamaño del block es igual al tamaño de la matriz, usted está usando el método tradicional.

Usando eigen En este caso usted usará matrices de eigen y sus funciones internas para calcular la operación apropiada. Por ejemplo, para calcular la matriz transpuesta sería simplemente

```
AT = A.transpose();
```

Como banderas, debe hacer las comparaciones sin usar optimización (-O0), y luego usando optimización (-O3).

NOTA IMPORTANTE: Al explorar los efectos de la optimización, tal vez deba implementar alguna operación extra sobre la matriz de resultado, como hacer la suma de todos los elementos y luego imprimir el resultado (fuera de la parte que se está midiendo), para que el compilador NO elimine los cálculos y usted obtenga una medida de desempeño errónea. Recuerde que usted va a medir el performance solamente de la parte que calcula la transpuesta o la multiplicación. El resto de operaciones (como la declaración y reserva de memoria, o la suma de los elementos) no será tenida en cuenta en la medida del performance, es decir, debe estar fuera de la función `code_to_be_measured` (ver ejemplo final).

NOTA IMPORTANTE 2: Para el cálculo directo y el cálculo por blocking use arreglos primitivos de C++ (preferiblemente modelados de forma unidimensional para que sean contiguos en memoria).

3 Estudio computacional

Usted va a estudiar el desempeño, medido en tiempo de CPU y en MFLOPS, de las dos técnicas mencionadas (blocking, eigen) cuando son aplicadas a los dos problemas mencionados (calcular la transpuesta y calcular la multiplicación de matrices). Adicionalmente, en el caso de blocking, deberá explorar la eficiencia en función del tamaño de los bloques. Por lo tanto, los dos estudios generales que debe hacer son:

3.1 Performance de la técnica de blocking en función del tamaño del bloque

Fije el tamaño de la matriz en $N = 8192$. Estudie el performance (tiempo de ejecución y MFLOPS) de la transpuesta y de la multiplicación de matrices en función del tamaño del bloque. Tome los tamaños del bloque como $N_b = 1, 2, 4, 8, 16, 32, 64, 128, 256, 512, 1024, 2048, 4096, 8192$. Haga una gráfica, para cada problema, del tiempo y de MFLOPS en función del tamaño del bloque. En total debe presentar las cuatro figuras que siguen:

1. Tiempo de CPU en función de N_b para la transpuesta.
2. MFLOPS (en lo posible normalizados con el peak performance del procesador) en función de N_b para la transpuesta.
3. Tiempo de CPU en función de N_b para la multiplicación de matrices.
4. MFLOPS (en lo posible normalizados con el peak performance del procesador) en función de N_b para la multiplicación de matrices.

Verifique si el óptimo tamaño del bloque que obtuvo es igual con y sin optimización. En el siguiente punto debe usar el óptimo que aplique en cada caso.

3.2 Performance en función del tamaño de la matriz, tanto para blocking como para eigen

En este caso usted variará el tamaño de la matriz. Tome $N = 2, 4, 8, 16, 32, 64, 128, 256, 512, 1024, 2048, 4096, 8192, 16384$. Fije el tamaño del bloque en el óptimo que encuentra en el estudio anterior. Para cada nivel de optimización, debe presentar las siguientes figuras (en total serían ocho figuras):

1. Tiempo de CPU en función de N para la transpuesta, usando el blocking (con tamaño fijo óptimo de bloque) y eigen.
2. MFLOPS (en lo posible normalizados con el peak performance del procesador) en función de N para la transpuesta, usando la implementación el blocking (con tamaño fijo de bloque), y eigen.
3. Tiempo de CPU en función de N para la multiplicación de matrices usando el blocking (con tamaño fijo de bloque) y eigen.
4. MFLOPS (en lo posible normalizados con el peak performance del procesador) en función de N para la multiplicación de matrices, usando el blocking (con tamaño fijo de bloque) y eigen.

4 Entregables

4.1 Informe

Su grupo deberá entregar un informe, tipo artículo (hecho en \LaTeX), que debe:

- Explicar el problema.
- Detallar la máquina en la que se realizaron las pruebas.
- Detallar las versiones de sistema operativo, compilador, librerías, etc.
- Indicar la experiencia en implementación, incluyendo los posibles problemas y la forma en la que la resolvieron.
- Los resultados y el **análisis** de la implementación y la solución de cada uno de los problemas descritos. Por ejemplo, debe indicar si el desempeño siempre es mejor o no con blocking, por qué, si el blocking siempre aumenta en desempeño o si en algún momento cae, que relación o influencia tiene la cache del procesador, etc.

4.2 Repositorio

La url del repositorio de entrega es: <https://classroom.github.com/g/m2CYJtVh>

El repositorio deberá reflejar las buenas prácticas de git (muchos pequeños commits cada uno afectando una parte concreta del código, debe tener commits de cada miembro del grupo, etc). El repositorio también debe contener el documento latex y en general lo necesario para reconstruir el informe. Los únicos archivos de tipo binario en el repositorio deben ser las figuras del reporte y el pdf del reporte mismo, por lo demás lo único que debe contener el repositorio son archivos de texto (los `.cpp`, el `.tex`, y el `Readme.txt`). El profesor descargará todo del repositorio. Nada deberá ser enviado por email (salvo las dudas). Los makefiles obligatorios servirán tanto para compilar los programas como para generar el reporte.

Automatice al máximo el estudio, idealmente leyendo argumentos desde la línea de comando o desde un archivo, y si lo desea usando un script de shell (denotado `run.sh` en el ejemplo).

La estructura del repositorio debe ser como se muestra a continuación:

```
.
|-- code
|   |-- Makefile
|   |-- multiplicacion_blocking.cpp
|   |-- multiplicacion_eigen.cpp
|   |-- run.sh
|   |-- transpuesta_blocking.cpp
|   |-- transpuesta_eigen.cpp
|-- Readme.txt
-- report
    |-- fig
    |   |-- mult-MFLOPS_vs_NB-00.pdf
```

```

| |-- mult-MFLOPS_vs_NB-03.pdf
| |-- mult-MFLOPS_vs_N-00.pdf
| |-- mult-MFLOPS_vs_N-03.pdf
| |-- mult-t_vs_N-00.pdf
| |-- mult-t_vs_N-03.pdf
| |-- trans-MFLOPS_vs_NB-00.pdf
| |-- trans-MFLOPS_vs_NB-03.pdf
| |-- trans-MFLOPS_vs_N-00.pdf
| |-- trans-MFLOPS_vs_N-03.pdf
| |-- trans-t_vs_N-00.pdf
| `-- trans-t_vs_N-03.pdf
|-- Makefile
|-- report.pdf
`-- report.tex

```

5 Ejemplo básico

El siguiente es un ejemplo del código que puede usar para calcular el performance de la transpuesta en la implementación sencilla. Puede usarlo como base para sus programas. Por ejemplo, para medir el performance usando blocking, deberá implementar ese algoritmo dentro de la función `code_to_be_measured`. Note que este es un código de C++ .

```

#include <iostream>
#include <cstdio>
#include <cstdlib>
#include "papi.h"

const int N = 1024;

int code_to_be_measured(const double * M, double * MT);

int main(int argc, char **argv)
{
    // Matrix declaration : Modeled as 1D array
    // Declare as pointers and ask for memory to use the heap
    double *A = new double [N*N], *AT = new double [N*N];

    // initialize matrices
    for (int ii = 0; ii < N; ++ii) {
        for (int jj = 0; jj < N; ++jj) {
            A[ii*N + jj] = ii + jj + 1;
            AT[ii*N + jj] = 0.0;
        }
    }

    // PAPI vars
    float real_time, proc_time, mflops;
    long long flpops;
    float ireal_time, iproc_time, imflops;
    long long iflpops;
    int retval;

    // PERFORMANCE MEASURE

    // start PAPI counters
    if((retval=PAPI_flops(&ireal_time,&iproc_time,&iflpops,&imflops)) < PAPI_OK)
    {
        printf("Could not initialise PAPI_flops \n");
        printf("Your platform may not support floating point operation event.\n");
        printf("retval: %d\n", retval);
        exit(1);
    }

    code_to_be_measured(A, AT);

    if((retval=PAPI_flops( &real_time, &proc_time, &flpops, &mflops))<PAPI_OK)
    {
        printf("retval: %d\n", retval);
        exit(1);
    }
}

```

```

    }

    printf("Real_time: %f Proc_time: %f Total flpops: %lld MFLOPS: %f\n",
           real_time, proc_time, flpops, mflops);

    // Do something here, like computing the average of the resulting matrix, to avoid the optimizer deleting the code
    printf("%e.15\n", AT[0][0]);

    delete [] A;
    delete [] AT;

    return 0;
}

int code_to_be_measured(const double * M, double * MT)
{
    // simple matrix transpose
    for (int ii = 0; ii < N; ++ii) {
        for (int jj = 0; jj < N; ++jj) {
            MT[ii*N + jj] = 2.3456*M[jj*N + ii]; // use temporal floating point operation to count mflops
        }
    }
    return 0;
}

```

El que sigue es un ejemplo similar pero usando eigen :

```

#include <iostream>
#include <cstdio>
#include <cstdlib>
#include <Eigen3/Eigen/Dense>
#include "papi.h"

const int N = 1024;

int code_to_be_measured(const Eigen::MatrixXd & M, Eigen::MatrixXd & MT);

int main(int argc, char **argv)
{
    // Matrix declaration
    Eigen::MatrixXd A(N, N), AT(N, N);

    // initialize matrices
    for (int ii = 0; ii < N; ++ii) {
        for (int jj = 0; jj < N; ++jj) {
            A(ii, jj) = ii + jj + 1;
            AT(ii, jj) = 0.0;
        }
    }

    // PAPI vars
    float real_time, proc_time, mflops;
    long long flpops;
    float ireal_time, iproc_time, imflops;
    long long iflpops;
    int retval;

    // PERFORMANCE MEASURE

    // start PAPI counters
    if((retval=PAPI_flops(&ireal_time,&iproc_time,&iflpops,&imflops)) < PAPI_OK)
    {
        printf("Could not initialise PAPI_flops \n");
        printf("Your platform may not support floating point operation event.\n");
        printf("retval: %d\n", retval);
        exit(1);
    }

    code_to_be_measured(A, AT);

    if((retval=PAPI_flops( &real_time, &proc_time, &flpops, &mflops))<PAPI_OK)
    {
        printf("retval: %d\n", retval);
        exit(1);
    }
}

```

```
printf("Real_time: %f Proc_time: %f Total flpops: %lld MFLOPS: %f\n",
      real_time, proc_time, flpops, mflops);

// Do something here, like computing the average of the resulting matrix, to avoid the optimizer deleting the code
printf("%.15e\n", AT[0][0])

return 0;
}

int code_to_be_measured(const Eigen::MatrixXd & M, Eigen::MatrixXd & MT)
{
    MT = 2.3456*M.transpose().eval();
    return 0;
}
```
