LIKWID

Like I Knew What I'm Doing

Peter Frank Perroni

November 26, 2015

Conjunto de ferramentas que:

- Monitora os contadores de performance do sistema.
- Força a afinidade de um programa com um ou vários cores.
- Exibe a hierarquia de memória e os cores.
- Faz micro-benchmarking do sistema.

* Suporta programas multi-thread.

Ferramentas mais utilizadas:

- **likwid-topology:** Exibe os cores e a hierarquia de memória do sistema.
- **likwid-bench:** Executa um micro-benhmarking no sistema.
- **likwid-perfctr:** Monitora os contadores de performance disponíveis nos processadores.
- **likwid-perfscore:** Exibe graficamente a evolução dos contadores de performance.

Ajustando variáveis de ambiente

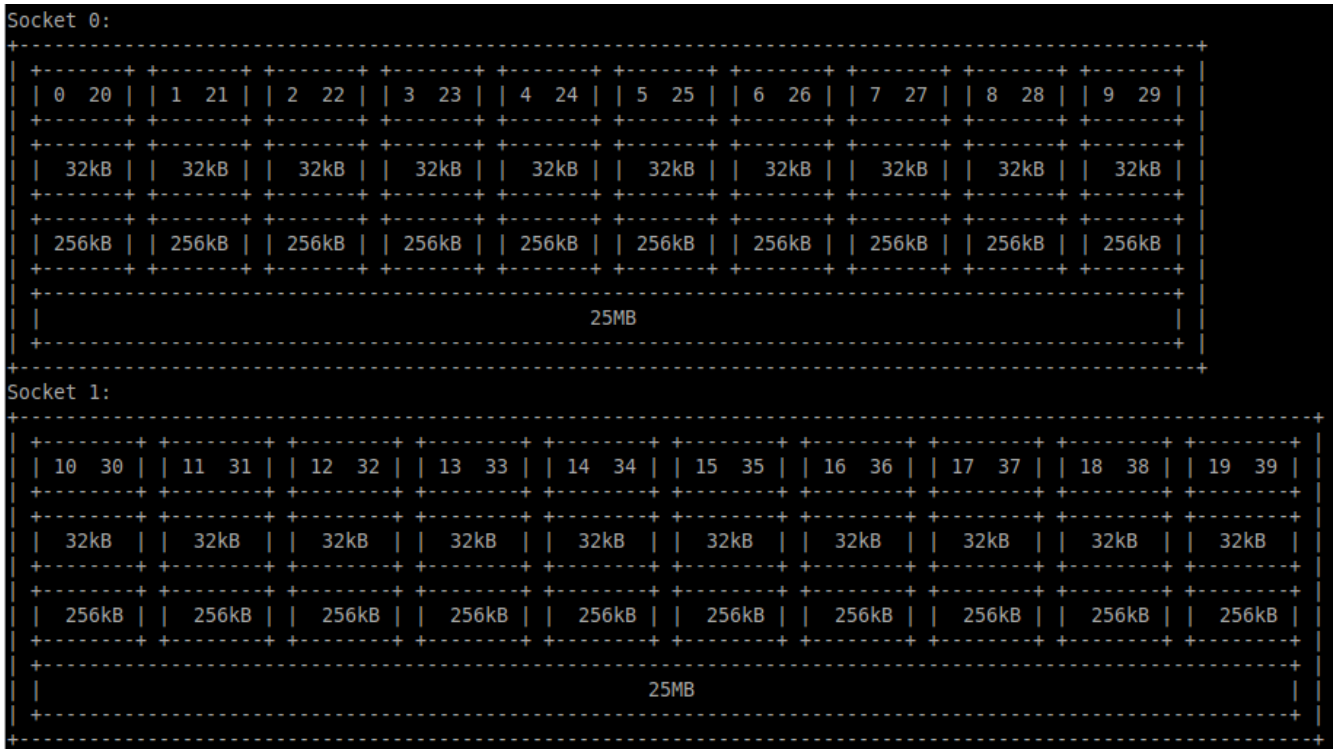
- Logue-se ao servidor latrappe
- Adicione ao final do arquivo `~/.bashrc`:

```
export PATH=/home/soft/likwid/bin:/home/soft/likwid/sbin:$PATH
export LD_LIBRARY_PATH=/home/soft/likwid/lib:$LD_LIBRARY_PATH
```

- Execute o comando:

```
$ source ~/.bashrc
```

likwid-topology -c -g



likwid-bench -a

Lista os benchmarks disponíveis.

likwid-bench -a

Lista os benchmarks disponíveis.

likwid-bench -i <iterações> -t <benchmark> -g <workgroups> -w <configuração>

- **Iterações:** Número de iterações a executar o benchmark.
- **Benchmark:** Benchmark a executar.
- **Workgroups** Número de grupos de trabalho a processar o benchmark.
- **Configuração:** `<socket>:<tamanho>:<#threads>` (1 config. por workgroup)

likwid-bench -a

Lista os benchmarks disponíveis.

likwid-bench -i <iterações> -t <benchmark> -g <workgroups> -w <configuração>

- **Iterações:** Número de iterações a executar o benchmark.
- **Benchmark:** Benchmark a executar.
- **Workgroups** Número de grupos de trabalho a processar o benchmark.
- **Configuração:** <socket>:<tamanho>:<#threads> (1 config. por workgroup)

```
likwid-bench -i 50000 -t sum -g 1 -w S0:10MB:6
```

likwid-bench -i 50000 -t sum_avx -g 1 -w S0:10MB:6

Compare com o benchmark anterior:

- Ciclos
- Tempo
- MFLOPS/s
- MBytes/s

likwid-bench -i 50000 -t sum_avx -g 1 -w S0:10MB:6

Compare com o benchmark anterior:

- Ciclos
- Tempo
- MFLOPS/s
- MBytes/s

likwid-bench -i 50000 -t triad -g 1 -w S0:10MB:6
likwid-bench -i 50000 -t triad_avx -g 1 -w S0:10MB:6

likwid-bench -i 50000 -t sum_avx -g 1 -w S0:10MB:6

Compare com o benchmark anterior:

- Ciclos
- Tempo
- MFLOPS/s
- MBytes/s

likwid-bench -i 50000 -t triad -g 1 -w S0:10MB:6
likwid-bench -i 50000 -t triad_avx -g 1 -w S0:10MB:6

$A(i) = B(i) + C(i) * D(i)$

- 3 Loads e 1 Store
- AVX se beneficia com este tipo de operação.
- Cópia de dados em `sum_avx` impacta na performance.

likwid-perfctr -e

Lista os contadores de performance e eventos disponíveis no sistema.

- Dependente de hardware.
- Arquiteturas diferentes possuem contadores e eventos distintos.

likwid-perfctr -e

Lista os contadores de performance e eventos disponíveis no sistema.

- Dependente de hardware.
- Arquiteturas diferentes possuem contadores e eventos distintos.

likwid-perfctr -a

Lista os grupos de performance disponíveis no sistema.

- Facilita a utilização dos contadores de performance.
- Os grupos resumem os dados coletados durante o monitoramento.
- Grupos também são dependentes de arquitetura.
  - O mesmo grupo em arquiteturas diferentes pode retornar contadores diferentes.

likwid-perfctr

Duas formas de utilizar:

- Executando diretamente sobre um código compilado.

likwid-perfctr

Duas formas de utilizar:

- Executando diretamente sobre um código compilado.

```
likwid-perfctr -C <core> -g <group> <program-to-monitor>
```

likwid-perfctr

Duas formas de utilizar:

- Executando diretamente sobre um código compilado.

```
likwid-perfctr -C <core> -g <group> <program-to-monitor>
```

- Instrumentando o código fonte com uma API específica.
  - Isola-se somente a porção que se deseja monitorar.

likwid-perfctr

Duas formas de utilizar:

- Executando diretamente sobre um código compilado.

```
likwid-perfctr -C <core> -g <group> <program-to-monitor>
```

- Instrumentando o código fonte com uma API específica.
  - Isola-se somente a porção que se deseja monitorar.

```
#include <likwid.h>
int main(int argc, char *argv[]) {
   LIKWID_MARKER_INIT;
   ...
   LIKWID_MARKER_START("marker-name");
   ...<the isolated code goes here>
   LIKWID_MARKER_STOP("marker-name");
   ...
   LIKWID_MARKER_CLOSE;
   ...
```

likwid-perfctr

Para o código instrumentado, o programa deve ser compilado com os parâmetros:

```
-DLIKWID_PERFMON -llikwid
```

E ao executar likwid-perfctr, o parâmetro `-m` deve ser adicionado.

likwid-perfctr

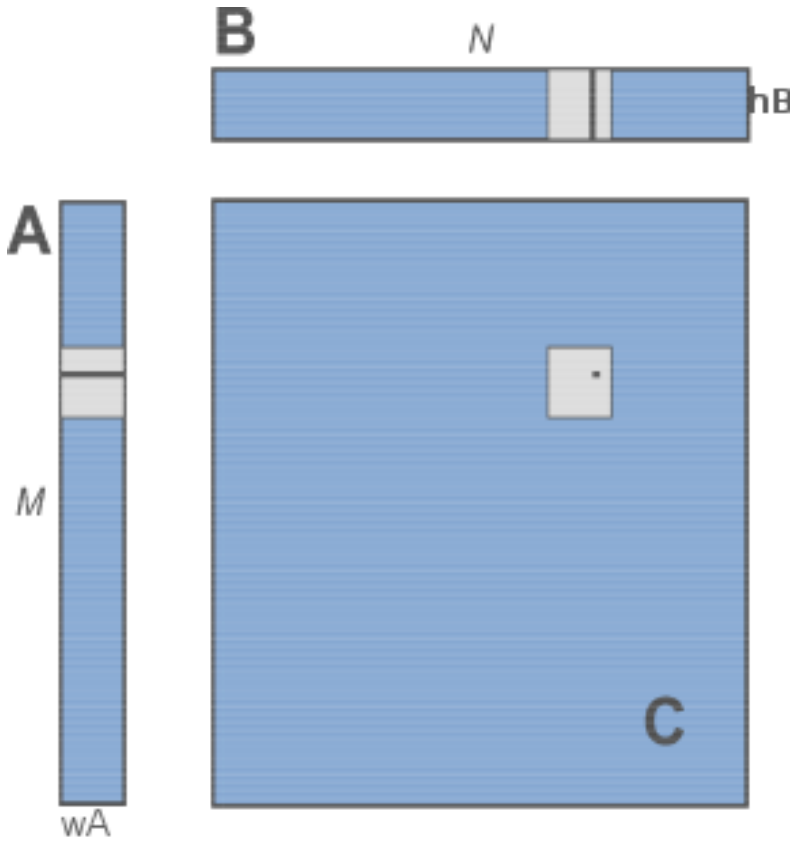
Para o código instrumentado, o programa deve ser compilado com os parâmetros:

```
-DLIKWID_PERFMON -llikwid
```

E ao executar likwid-perfctr, o parâmetro `-m` deve ser adicionado.

```
likwid-perfctr -C <core> -g <group> -m <program-to-monitor>
```

Multiplicação de matrizes



Multiplicação Simples

```
for(int c, r=0; r < M; r++){     // For each line of matrix A.
   for(c=0; c < N; c++){         // For each column of matrix B.
      for(sum=0, i=0; i < WA; i++){ // C[r, c] = A[r] x B[c].
         sum += A[r * WA + i] * B[i * N + c];
      }
      C[r * N + c] = sum;
   }
}
```

Multiplicação Simples

Ainda na latrappe:

```
$ mkdir likwid
$ cd likwid
$ wget http://www.inf.ufpr.br/pfperroni/CI164.tar.gz
$ tar -xzvf CI164.tar.gz
$ cd CI164
$ make
```

Multiplicação Simples

Ainda na latrappe:

```
$ mkdir likwid
$ cd likwid
$ wget http://www.inf.ufpr.br/pfperroni/CI164.tar.gz
$ tar -xzvf CI164.tar.gz
$ cd CI164
$ make
$ ./matrix_mult simple 1024 1024 1024
```

Packed double vs. Scalar double

- **Scalar:** Apenas os bits menos significantes dos registradores de 128 bits são utilizados.
  - Processamento sequencial.
- **Packed:** Todos os bits dos registradores de 128 bits são utilizados.
  - Processamento paralelo.

Packed double vs. Scalar double

- **Scalar:** Apenas os bits menos significantes dos registradores de 128 bits são utilizados.
  - Processamento sequencial.
- **Packed:** Todos os bits dos registradores de 128 bits são utilizados.
  - Processamento paralelo.

SIMD vs. AVX

- **SIMD:** Single Instruction Multiple Data, realiza operações em paralelo sobre os registradores SIMD de 128 bits.
- **AVX:** Advanced Vector Extensions, expande o SIMD para 256 bits.

Multiplicação AVX

```
for(int c, r=0; r < M; r++){
   for(c=0; c < N; c++){
      cAligned[0] = cAligned[1] = cAligned[2] = cAligned[3] = 0;
      for(i=0; i < WA4; i+=4){
         for(j=0; j < 4; j++){ // Align the data (256 bits).
            aAligned[j] = A[r * WA + i + j];
            bAligned[j] = B[(i+j) * N + c];
         }
         cAligned += aAligned * bAligned;
      }
      sum = cAligned[0] + cAligned[1]
            + cAligned[2] + cAligned[3];
      for(; i < WA; i++){ // Sum the remaining values.
         sum += A[r * WA + i] * B[i * N + c];
      }
      C[r * N + c] = sum;
   }
}
```

Parâmetros de compilação:

- -DLIKWID_PERFMON: Habilita a instrumentação LIKWID.
- -O: Nível de otimização desejado (0=nenhum, 3=máximo).
- -march=native: Compila especificamente para a arquitetura local.
- -ftree-vectorizer-verbose: Nível de debug da vetorização.

Parâmetros de compilação:

- -DLIKWID_PERFMON: Habilita a instrumentação LIKWID.
- -O: Nível de otimização desejado (0=nenhum, 3=máximo).
- -march=native: Compila especificamente para a arquitetura local.
- -ftree-vectorizer-verbose: Nível de debug da vetorização.

Abra o Makefile, comente a linha de otimização zero e descomente a de otimização máxima.

```
$ make clean
$ make
$ likwid-perfctr -C <#core> -g FLOPS_AVX -m ./matrix_mult simple
1024 1024 1024
$ likwid-perfctr -C <#core> -g FLOPS_AVX -m ./matrix_mult avx
1024 1024 1024
```

Parâmetros de compilação:

- -DLIKWID_PERFMON: Habilita a instrumentação LIKWID.
- -O: Nível de otimização desejado (0=nenhum, 3=máximo).
- -march=native: Compila especificamente para a arquitetura local.
- -ftree-vectorizer-verbose: Nível de debug da vetorização.

Abra o Makefile, comente a linha de otimização zero e descomente a de otimização máxima.

```
$ make clean
$ make
$ likwid-perfctr -C <#core> -g FLOPS_AVX -m ./matrix_mult simple
1024 1024 1024
$ likwid-perfctr -C <#core> -g FLOPS_AVX -m ./matrix_mult avx
1024 1024 1024
```

SIMD_FP_256_PACKED_DOUBLE só é utilizado com o código avx.

```
$ exit # sai da latrappe.
```

→ Baixe e descompacte a ferramenta no seu home.

```
$ cd likwid/CI164
$ make
$ mv matrix_mult matrix_mult00
```

Altere o Makefile para otimização máxima e recompile.

```
$ make clean
$ make
$ mv matrix_mult matrix_mult03
```

```
$ exit # sai da latrappe.
```

→ Baixe e descompacte a ferramenta no seu home.

```
$ cd likwid/CI164
$ make
$ mv matrix_mult matrix_mult00
```

Altere o Makefile para otimização máxima e recompile.

```
$ make clean
$ make
$ mv matrix_mult matrix_mult03
```

Execute:

```
$ likwid-perfctr -C <#core> -g DATA -m ./matrix_mult00 simple 1024 1024 1024
$ likwid-perfctr -C <#core> -g DATA -m ./matrix_mult03 simple 1024 1024 1024

$ likwid-perfctr -C <#core> -g DATA -m ./matrix_mult03 avx 1024 1024 1024
```

```
$ exit # sai da latrappe.
```

→ Baixe e descompacte a ferramenta no seu home.

```
$ cd likwid/CI164
$ make
$ mv matrix_mult matrix_mult00
```

Altere o Makefile para otimização máxima e recompile.

```
$ make clean
$ make
$ mv matrix_mult matrix_mult03
```

Execute:

```
$ likwid-perfctr -C <#core> -g DATA -m ./matrix_mult00 simple 1024 1024 1024
$ likwid-perfctr -C <#core> -g DATA -m ./matrix_mult03 simple 1024 1024 1024

$ likwid-perfctr -C <#core> -g DATA -m ./matrix_mult03 avx 1024 1024 1024
```

- Baixo load:store no binário sem -O3.
- Load-to-store da multiplicação simples foi bem otimizado pelo compilador.
- O custo do alinhamento da memória para o cálculo em AVX foi alto (baixo load-to-store).

Exercício 1

Implemente na ferramenta `matrix_mult` uma multiplicação de Vetor x Matriz nos modos:

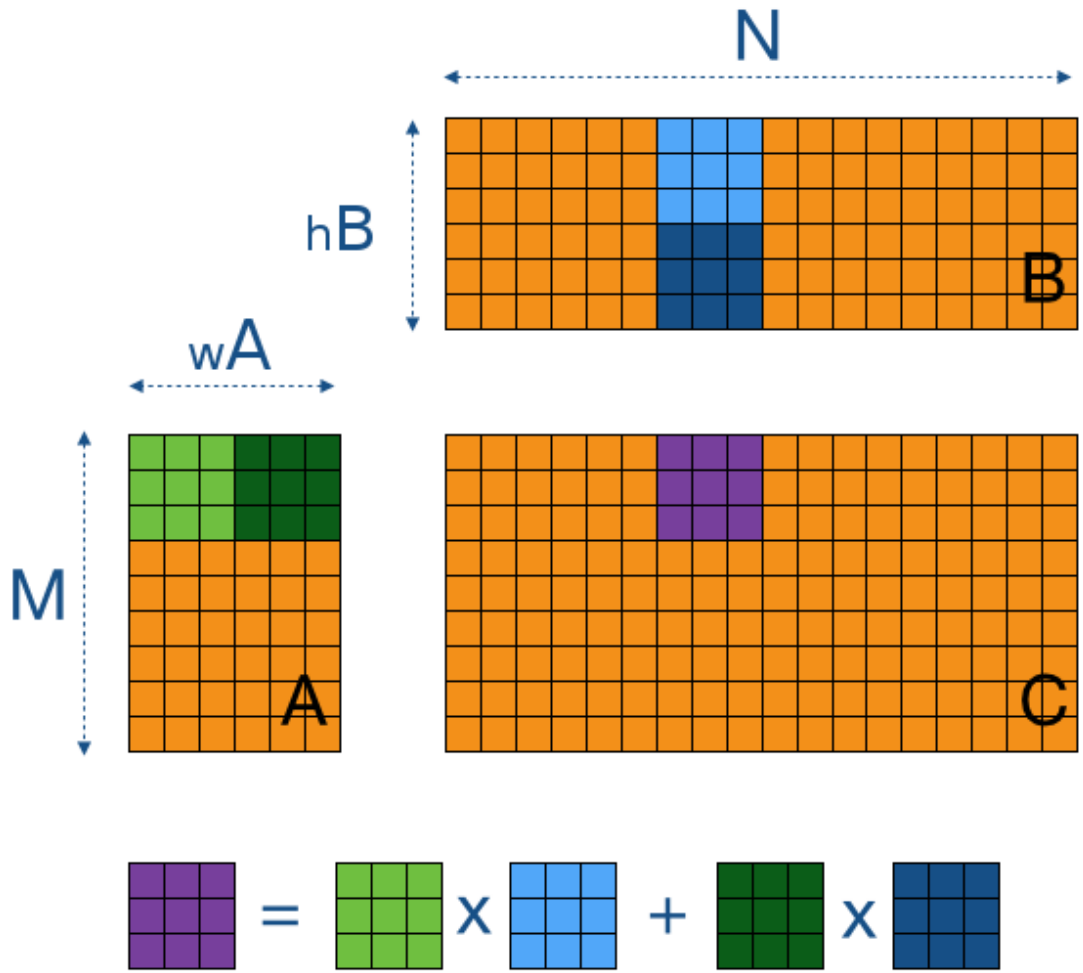
- `vector`: Vetor x Matriz Simples
- `vector-avx`: Vetor x Matriz utilizando AVX

Siga o mesmo formato de código da ferramenta, adicionando as duas novas funções como uma opção extra.

Utilize o `likwid-perfctr` para avaliar as diferenças entre os dois algoritmos (`vector` e `vector-avx`).

- O que mais chamou atenção entre as diferenças encontradas?
- Quais os motivos que levaram a estas diferenças?

Multiplicação de matrizes por blocos



Multiplicação por Blocos

```
for(rb=0; rb < M; rb+=BlkSize){
   for(cb=0; cb < N; cb+=BlkSize){
        // Iterate through the blocks.
        for(Step=0; Step < WA; Step+=BlkSize){
           m = min(BlkSize, M-rb); // Block rows of A.
           n = min(BlkSize, N-cb); // Block columns of B.
           S = min(BlkSize, WA-Step); // Block columns of A.
           // For each row of each block of A.
           for(r=0; r < m; r++){
              // For each column of each block of B.
              for(c=0; c < n; c++){
                 // C[block(rb, cb)] += A[block(rb, Step)]
                                       * B[block(Step, cb)].
                 for(sum=0, s=0; s < S; s++){
                    sum += A[(rb+r)*WA + Step + s]
                         * B[(Step+s)*N + cb + c];
                 }
                 C[(rb+r)*N + cb + c] += sum;
           }
        }
     }
   }
}
```

```
$ likwid-perfctr -C <#core> -g L2CACHE -m ./matrix_mult00 simple 1024 1024 1024

$ likwid-perfctr -C <#core> -g L2CACHE -m ./matrix_mult00 block 1024 1024 1024
```

```
$ likwid-perfctr -C <#core> -g L2CACHE -m ./matrix_mult00 simple 1024 1024 1024
```

```
$ likwid-perfctr -C <#core> -g L2CACHE -m ./matrix_mult00 block 1024 1024 1024
```

Compare as requisições para L2, L2 cache miss e tempo de processamento.

Multiplicação por Blocos Transpostos

```
for(rb=0; rb < M; rb+=BlkSize){
   for(cb=0; cb < N; cb+=BlkSize){
      for(Step=0; Step < WA; Step+=BlkSize){
         m = min(BlkSize, M-rb);
         n = min(BlkSize, N-cb);
         S = min(BlkSize, WA-Step);
         S4 = (S / 4) * 4;
         // Transpose the B block.
         for(r=0; r < m; r++){
            for(c=0; c < n; c++){
               bt[c * BlkSize + r] = B[(Step+r)*N+cb+c];
            }
         }
         for(r=0; r < m; r++){
            for(c=0; c < n; c++){
               p1 = (rb+r)*WA + Step;
               for(sum=0, s=0; s < S4; s+=4, p1+=4){
                  // Unroll (4 packed doubles).
                  sum += A[p1] * bt[c*BlkSize+s]
                       + A[p1+1] * bt[c*BlkSize+s+1]
                       + A[p1+2] * bt[c*BlkSize+s+2]
                       + A[p1+3] * bt[c*BlkSize+s+3];
               }
               for(; s < S; s++) sum += A[p1++] * bt[c*BlkSize+s];
               C[(rb+r)*N + cb + c] += sum;
            }
         }
      }
```

```
$ likwid-perfctr -C <#core> -g L2 -m ./matrix_mult00 simple 1024 1024 1024
$ likwid-perfctr -C <#core> -g L2 -m ./matrix_mult00 block-transpose 1024 1024 1024
```

```
$ likwid-perfctr -C <#core> -g L2 -m ./matrix_mult00 simple 1024 1024 1024
$ likwid-perfctr -C <#core> -g L2 -m ./matrix_mult00 block-transpose 1024 1024 1024
```

Compare cache line refill e volume de dados.

```
$ likwid-perfctr -C <#core> -g L2 -m ./matrix_mult00 simple 1024 1024 1024
$ likwid-perfctr -C <#core> -g L2 -m ./matrix_mult00 block-transpose 1024 1024 1024
```

Compare cache line refill e volume de dados.

```
$ likwid-perfctr -C <#core> -g L3 -m ./matrix_mult00 simple 1024 1024 1024
$ likwid-perfctr -C <#core> -g L3 -m ./matrix_mult00 block-transpose 1024 1024 1024
```

```
$ likwid-perfctr -C <#core> -g L2 -m ./matrix_mult00 simple 1024 1024 1024
$ likwid-perfctr -C <#core> -g L2 -m ./matrix_mult00 block-transpose 1024 1024 1024
```

Compare cache line refill e volume de dados.

```
$ likwid-perfctr -C <#core> -g L3 -m ./matrix_mult00 simple 1024 1024 1024
$ likwid-perfctr -C <#core> -g L3 -m ./matrix_mult00 block-transpose 1024 1024 1024
```

Compare cache line refill, volume de dados e largura de banda.

```
$ likwid-perfctr -C <#core> -g L2 -m ./matrix_mult00 simple 1024 1024 1024
$ likwid-perfctr -C <#core> -g L2 -m ./matrix_mult00 block-transpose 1024 1024 1024
```

Compare cache line refill e volume de dados.

```
$ likwid-perfctr -C <#core> -g L3 -m ./matrix_mult00 simple 1024 1024 1024
$ likwid-perfctr -C <#core> -g L3 -m ./matrix_mult00 block-transpose 1024 1024 1024
```

Compare cache line refill, volume de dados e largura de banda.

```
$ likwid-perfctr -C <#core> -g L3 -m ./matrix_mult03 block-avx 1024 1024 1024
$ likwid-perfctr -C <#core> -g L3 -m ./matrix_mult03 block 1024 1024 1024
$ likwid-perfctr -C <#core> -g L3 -m ./matrix_mult03 block-transpose 1024 1024 1024
```

```
$ likwid-perfctr -C <#core> -g L2 -m ./matrix_mult00 simple 1024 1024 1024
$ likwid-perfctr -C <#core> -g L2 -m ./matrix_mult00 block-transpose 1024 1024 1024
```

Compare cache line refill e volume de dados.

```
$ likwid-perfctr -C <#core> -g L3 -m ./matrix_mult00 simple 1024 1024 1024
$ likwid-perfctr -C <#core> -g L3 -m ./matrix_mult00 block-transpose 1024 1024 1024
```

Compare cache line refill, volume de dados e largura de banda.

```
$ likwid-perfctr -C <#core> -g L3 -m ./matrix_mult03 block-avx 1024 1024 1024
$ likwid-perfctr -C <#core> -g L3 -m ./matrix_mult03 block 1024 1024 1024
$ likwid-perfctr -C <#core> -g L3 -m ./matrix_mult03 block-transpose 1024 1024 1024
```

Alinhamento no AVX acarretou mais trocas de dados dos caches (volume de dados), e consequentemente, reduziu a performance.

```
$ likwid-perfctr -C <#core> -g L2 -m ./matrix_mult00 simple 1024 1024 1024
$ likwid-perfctr -C <#core> -g L2 -m ./matrix_mult00 block-transpose 1024 1024 1024
```

Compare cache line refill e volume de dados.

```
$ likwid-perfctr -C <#core> -g L3 -m ./matrix_mult00 simple 1024 1024 1024
$ likwid-perfctr -C <#core> -g L3 -m ./matrix_mult00 block-transpose 1024 1024 1024
```

Compare cache line refill, volume de dados e largura de banda.

```
$ likwid-perfctr -C <#core> -g L3 -m ./matrix_mult03 block-avx 1024 1024 1024
$ likwid-perfctr -C <#core> -g L3 -m ./matrix_mult03 block 1024 1024 1024
$ likwid-perfctr -C <#core> -g L3 -m ./matrix_mult03 block-transpose 1024 1024 1024
```

Alinhamento no AVX acarretou mais trocas de dados dos caches (volume de dados), e consequentemente, reduziu a performance.

```
$ likwid-perfctr -C <#core> -g L2 -m ./matrix_mult03 block-transpose 1024 1024 1024

$ likwid-perfctr -C <#core> -g L2 -m ./matrix_mult03 block-avx 1024 1024 1024
```

Compare cache refill e volume de dados.

Exercício 2

Implemente na ferramenta `matrix_mult` uma multiplicação de Vetor x Matriz nos modos:

- `vector-block`: Vetor x Matriz por Blocos
- `vector-block-transpose`: Vetor x Matriz por Blocos Transpostos

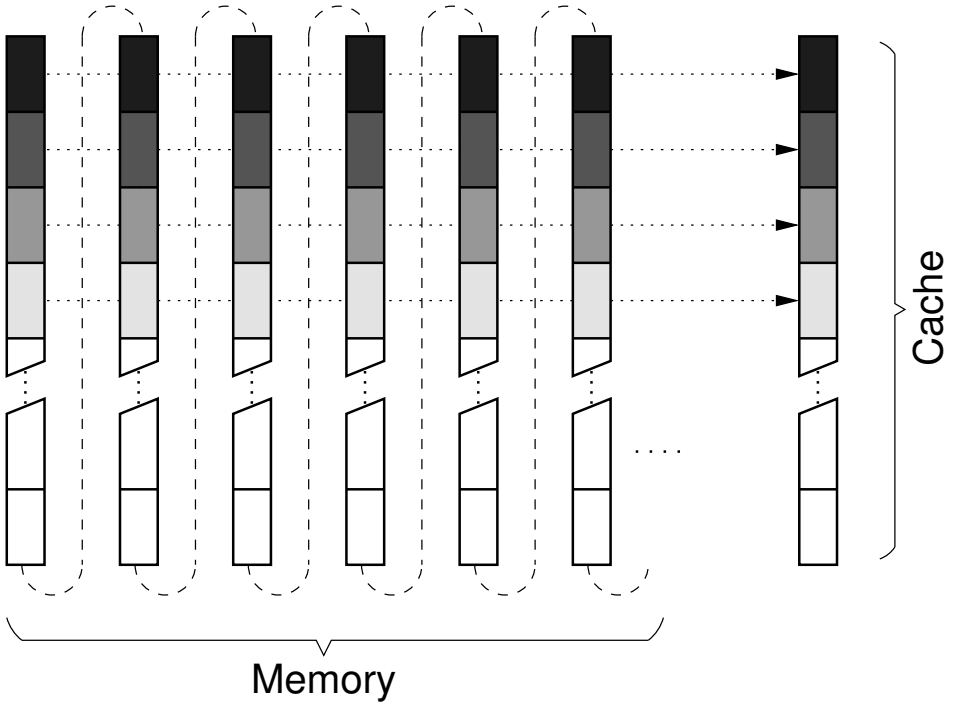
Siga o mesmo formato de código da ferramenta, adicionando as duas novas funções como uma opção extra.

Utilize o `likwid-perfctr` para avaliar as diferenças entre os quatro algoritmos desenvolvidos (`vector`, `vector-avx`, `vector-block` e `vector-block-transpose`).

- O que mais chamou atenção entre as diferenças encontradas?
- Quais os motivos que levaram a estas diferenças?
- Qual o melhor algoritmo para este problema e por quê?

Cache Thrashing

```
$ likwid-topology -c
  ...
  Cache line size:  64
  ...
```



Como evitar?

Como evitar?

Utilizar tamanho de colunas não múltiplos de cache line size.

```
$ likwid-perfctr -C <#core> -g L3 -m ./matrix_mult03 simple 1024
```

1025 1024

Como evitar?

Utilizar tamanho de colunas não múltiplos de cache line size.

```
$ likwid-perfctr -C <#core> -g L3 -m ./matrix_mult03 simple 1024
```

1025 1024

Compare com:

```
$ likwid-perfctr -C <#core> -g L3 -m ./matrix_mult03 simple 1024
```

1024 1024

O que você pode perceber com relação ao tráfego de dados?

likwid-perfscope

Compare visualmente qual dos métodos de multiplicação abaixo obteve melhor reaproveitamento dos dados em cada nível de cache.

```
$ likwid-perfscope -t 1s -C <#core> -g L2 ./matrix_mult03 simple
2048 2048 2048
$ likwid-perfscope -t 1s -C <#core> -g L3 ./matrix_mult03 simple
2048 2048 2048

$ likwid-perfscope -t 1s -C <#core> -g L2 ./matrix_mult03 block
2048 2048 2048
$ likwid-perfscope -t 1s -C <#core> -g L3 ./matrix_mult03 block
2048 2048 2048

$ likwid-perfscope -t 1s -C <#core> -g L2 ./matrix_mult03
block-transpose 2048 2048 2048
$ likwid-perfscope -t 1s -C <#core> -g L3 ./matrix_mult03
block-transpose 2048 2048 2048
```

Template para Plotagens

```
$ wget http://www.inf.ufpr.br/pfperroni/CI164-plot.tar.gz

$ tar -xzvf CI164-plot.tar.gz

$ gnuplot -p graph-template.plot
```

Realizando Experimentos
Matriz x Matriz

- Escolha 3 medidas de desempenho.
- Execute `likwid-perfctr` sobre `matrix_mult03` com as multiplicações `simple` e `block-transpose` para as seguintes dimensões:
  - 64x64x64, 512x512x512, 1024x1024x1024, 2048x2048x2048
  - 64x65x64, 512x513x512, 1024x1025x1024, 2048x2049x2048
- Colete os valores de desempenho de cada medida escolhida e salve-os em uma cópia do arquivo `graph-template.dat`.
  - Siga o examplo do próprio template.
  - 1 arquivo para cada medida (graph-medida_matriz-n.dat, $n = 1..3$).
- Ajuste uma cópia do arquivo `graph-template.plot` para cada medida selecionada (graph-medida_matriz-n.plot, $n = 1..3$).
- Plote cada um dos 3 gráficos com o comando:

```
$ gnuplot -p <graph-medida_matriz-n.plot>
```

Realizando Experimentos
Vetor x Matriz

- Escolha 3 medidas de desempenho.
- Execute `likwid-perfctr` sobre `matrix_mult03` com as multiplicações `vector` e `vector-block-transpose` para as seguintes dimensões:
  - 1x64x64, 1x512x512, 1x1024x1024, 1x2048x2048
- Colete os valores de desempenho de cada medida escolhida e salve-os em uma cópia do arquivo `graph-template.dat`.
  - 1 arquivo para cada medida (graph-medida_vetor-n.dat, $n = 1..3$).
- Ajuste uma cópia do arquivo `graph-template.plot` para cada medida selecionada (graph-medida_vetor-n.plot, $n = 1..3$).
- Plote cada um dos 3 gráficos com o comando:

```
$ gnuplot -p <graph-medida_vetor-n.plot>
```

- O que mais chamou atenção entre as diferenças encontradas nos gráficos?
- Quais os motivos que levaram a estas diferenças?
- Há algum tamanho de matriz onde as diferenças se tornam mais evidentes?
    - Qual a razão numérica desta diferença no melhor e no pior algoritmo?

Link Interessante

http://www.hpc.ut.ee/dokumendid/ips_xe_2015/vtune_amplifier_xe/documentation/en/help/reference/snbep/index.htm