Usage of STempEL

STempEL has 2 subcommands (gen and bench) used to generate a C like stencil kernel, used by kerncraft to analyse the stencil, and to create the benchmark code.

stempel gen accept several parameters:

- number of dimensions desired, by specifying the -D flag (i.e. -D 2)
- radius of the stencil, by specifying the -r flag (i.e. -r 2)
- kind of the stencil, that can be star or box, by using the -k flag (i.e. -k star). When not specified, it defaults to star
- type of the coefficients, that can be constant or variable, by specifying the -C flag (i.e. -C variable). When not specified, it defaults to constant
    - in case of variable coefficients, the dimension that stores the coefficients can be selected through the -d flag (i.e. -d 1). All the coefficients are packed into an array (e.g. W[M][N][2]) and this flag allows to modify how the array is built: -d 2 means W[M][N][2] -> W[M][2][N]
- data type of your computation, either double or float, by specifying the -t flag (i.e. -t float). When not specified, it defaults to double
- classification of the stencil with respect to its weighting factors. The possible, mutually exclusive, choices are:
    - isotropic, i.e. the coefficients do not depend on the direction, by passing the flag -i
    - heterogeneous, i.e. the weighting factors expose no simmetry (a different coefficient for each direction), by passing the flag -e
    - homogeneous, the only coefficient is a scalar, by passing the flag -o
    - point-symmetric, i.e. the weighting factors are simmetric to the origin, by passing the flag -p
- whether to store, by passing the --store flag (specifying the name of the file), or simply print to screen the generated stencil

An example of command line to generate a 3D radius 2 star point-symmetric stencil, with variable coefficients, stored in its first dimension, and float as data type is:

stempel gen -D 3 -r 2 -k star -C variable -p -t float -d 1

The output is:

```
float a[M][N][P];
float b[M][N][P];
float W[7][M][N][P];

for(int k=2; k < M-2; k++){
for(int j=2; j < N-2; j++){
for(int i=2; i < P-2; i++){
b[k][j][i] = W[0][k][j][i] * a[k][j][i]
+ W[1][k][j][i] * (a[k][j][i-1] + a[k][j][i+1])
+ W[2][k][j][i] * (a[k-1][j][i] + a[k+1][j][i])
+ W[3][k][j][i] * (a[k][j-1][i] + a[k][j+1][i])
+ W[4][k][j][i] * (a[k][j][i-2] + a[k][j][i+2])
+ W[5][k][j][i] * (a[k-2][j][i] + a[k+2][j][i])
+ W[6][k][j][i] * (a[k][j-2][i] + a[k][j+2][i])
;
}
}
}
```

In case we do not pass the datatype and we choose the second dimension to store the coefficients, we run the following command:
stempel gen -D 3 -r 2 -k star -C variable -p -d 2 --store stencil.c

and get this output saved in the file stencil.c:

```
double a[M][N][P];
double b[M][N][P];
double W[M][7][N][P];

for(int k=2; k < M-2; k++){
for(int j=2; j < N-2; j++){
for(int i=2; i < P-2; i++){
b[k][j][i] = W[k][0][j][i] * a[k][j][i]
+ W[k][1][j][i] * (a[k][j][i-1] + a[k][j][i+1])
+ W[k][2][j][i] * (a[k-1][j][i] + a[k+1][j][i])
+ W[k][3][j][i] * (a[k][j-1][i] + a[k][j+1][i])
+ W[k][4][j][i] * (a[k][j][i-2] + a[k][j][i+2])
+ W[k][5][j][i] * (a[k-2][j][i] + a[k+2][j][i])
+ W[k][6][j][i] * (a[k][j-2][i] + a[k][j+2][i])
;
}
}
}
```

The ouput of the generator produces a kernel accepted by kerncraft, thus allowing the analysis of the stencil an the modeling of its performance.

After the modeling we can pass to generate the benchmark code out of the kernel. In order to do so, we use the bench subcommand provided by stempel.

stempel bench accept several parameters:

- file containing the stencil specification (the one previously created using the --store flag)
- machine file containing the speciication of the architecture (same as kerncraft)
- blocking version or not, by passing the -b flag (the blocking is on the innermost loop when 2D and middle loop when 3D)
- whether to store, by passing the --store flag. It will generate a file called inputfilename_compilable.c and kernel.c, containing the main and the kernel function respectively.

The size of each dimension will be accepted by command line, when running the executable generated compiling the source code produced by stempel bench.

The command:
stempel bench stencil.c -m Intel_Xeon_CPU_E5-2640_v4_mod2.yml

produces the following output, saved in stencil_compilable.c:

```
#include <stdlib.h>
#include <math.h>

#include "timing.h"
#include "kerncraft.h"
#include "kernel.c"
#ifdef LIKWID_PERFMON
#include <likwid.h>
#endif

void* aligned_malloc(size_t, size_t);
extern int var_false;
void dummy(double *);
extern void kernel_loop(double *a, double *b, double *W, int M, int N, int P);
int main(int argc, char **argv)
{

  #ifdef LIKWID_PERFMON
```

```
LIKWID_MARKER_INIT;
#endif


#ifdef LIKWID_PERFMON
#pragma omp parallel
{
 LIKWID_MARKER_THREADINIT;
}
#endif

if (argc != 4)
{
 printf("Wrong number of arguments. Usage:\n%s size size size ", argv[0]);
 return(0);
}

int P = atoi(argv[3]);
int N = atoi(argv[2]);
int M = atoi(argv[1]);
double *a = aligned_malloc((sizeof(double)) * ((M * N) * P), 32);
for (int i = 0; i < ((M * N) * P); ++i)
 a[i] = rand() / ((double ) RAND_MAX);

double *b = aligned_malloc((sizeof(double)) * ((M * N) * P), 32);
for (int i = 0; i < ((M * N) * P); ++i)
 b[i] = rand() / ((double ) RAND_MAX);

double *W = aligned_malloc((sizeof(double)) * (((M * 7) * N) * P), 32);
for (int i = 0; i < (((M * 7) * N) * P); ++i)
 W[i] = (rand() / ((double ) RAND_MAX)) / 9.0;

int repeat = 1;
double runtime = 0.0;
double wct_start;
double wct_end;
double cput_start;
double cput_end;
double *tmp;

#ifdef LIKWID_PERFMON
#pragma omp parallel
{
 LIKWID_MARKER_START("Sweep");
}
#endif

while (runtime < 0.5)
{
 timing(&wct_start, &cput_start);
 for (int n = 0; n < repeat; ++n)
 {
  kernel_loop(a, b, W, M, N, P);
  tmp = a;
  a = b;
  b = tmp;
 }

 timing(&wct_end, &cput_end);
 runtime = wct_end - wct_start;
 repeat *= 2;
}
```

```
  #ifdef LIKWID_PERFMON
  #pragma omp parallel
  {
    LIKWID_MARKER_STOP("Sweep");
  }
  #endif

  repeat /= 2;
  printf("Performance in mlup/s: %lf\n", (((double ) repeat) * ((((double ) (M - 4)) * ((double ) (N - 4))) * ((double ) (P - 4)))) /
(runtime * 1000000.));
  printf("size: %d    time: %lf    iter: %d    mlup/s: %lf\n", (M * N * P), runtime, repeat, (((double ) repeat) * ((((double ) (M -
4)) * ((double ) (N - 4))) * ((double ) (P - 4)))) / (runtime * 1000000.));
  double total = 0.0;
  for (int k = 2; k < (M - 2); k++)
  {
    for (int j = 2; j < (N - 2); j++)
    {
      for (int i = 2; i < (P - 2); i++)
      {
        total = total + (a[(i + (j * P)) + (k * (P * N))] - b[(i + (j * P)) + (k * (P * N))]);
      }

    }

  }

  printf("diff(a-b): %lf\n", total);

  #ifdef LIKWID_PERFMON
  LIKWID_MARKER_CLOSE;
  #endif

}
```

and the file kernel.c:

```
#ifndef min
#define min( a, b ) ( ((a) < (b)) ? (a) : (b) )
#endif
void kernel_loop(double *a, double *b, double *W, int M, int N, int P)
{
  #pragma omp parallel for schedule(runtime)
  for (int k = 2; k < (M - 2); k++)
  {
    for (int j = 2; j < (N - 2); j++)
    {
      for (int i = 2; i < (P - 2); i++)
      {
        b[(i + (j * P)) + (k * (P * N))] = ((((((W[((i + (j * P)) + (0 * (P * N))) + (k * ((P * N) * 7))] * a[(i + (j * P)) + (k * (P * N))]) +
(W[((i + (j * P)) + (1 * (P * N))) + (k * ((P * N) * 7))] * (a[((i - 1) + (j * P)) + (k * (P * N))] + a[((i + 1) + (j * P)) + (k * (P * N))]))) +
(W[((i + (j * P)) + (2 * (P * N))) + (k * ((P * N) * 7))] * (a[(i + (j * P)) + ((k - 1) * (P * N))] + a[(i + (j * P)) + ((k + 1) * (P * N))]))) +
(W[((i + (j * P)) + (3 * (P * N))) + (k * ((P * N) * 7))] * (a[(i + ((j - 1) * P)) + (k * (P * N))] + a[(i + ((j + 1) * P)) + (k * (P * N))]))) +
(W[((i + (j * P)) + (4 * (P * N))) + (k * ((P * N) * 7))] * (a[((i - 2) + (j * P)) + (k * (P * N))] + a[((i + 2) + (j * P)) + (k * (P * N))]))) +
(W[((i + (j * P)) + (5 * (P * N))) + (k * ((P * N) * 7))] * (a[(i + (j * P)) + ((k - 2) * (P * N))] + a[(i + (j * P)) + ((k + 2) * (P * N))]))) +
(W[((i + (j * P)) + (6 * (P * N))) + (k * ((P * N) * 7))] * (a[(i + ((j - 2) * P)) + (k * (P * N))] + a[(i + ((j + 2) * P)) + (k * (P * N))]));
      }

    }

  }

}
```

It is possible to generate a blocked version of the code, using the following command line:
stempel bench stencil.c -m Intel_Xeon_CPU_E5-2640_v4_mod2.yml -b

and obtaining this stencil_compilable.c:

```
#include <stdlib.h>
#include <math.h>

#include "timing.h"
#include "kerncraft.h"
#include "kernel.c"
#ifdef LIKWID_PERFMON
#include <likwid.h>
#endif

void* aligned_malloc(size_t, size_t);
extern int var_false;
void dummy(double *);
extern void kernel_loop(double *a, double *b, double *W, int M, int N, int P, int block_factor);
int main(int argc, char **argv)
{

 #ifdef LIKWID_PERFMON
 LIKWID_MARKER_INIT;
 #endif


 #ifdef LIKWID_PERFMON
 #pragma omp parallel
 {
  LIKWID_MARKER_THREADINIT;
 }
 #endif

 if (argc != 5)
 {
  printf("Wrong number of arguments. Usage:\n%s size size size blocking", argv[0]);
  return(0);
 }

 int block_factor = atoi(argv[4]);
 int P = atoi(argv[3]);
 int N = atoi(argv[2]);
 int M = atoi(argv[1]);
 double *a = aligned_malloc((sizeof(double)) * ((M * N) * P), 32);
 for (int i = 0; i < ((M * N) * P); ++i)
  a[i] = rand() / ((double ) RAND_MAX);

 double *b = aligned_malloc((sizeof(double)) * ((M * N) * P), 32);
 for (int i = 0; i < ((M * N) * P); ++i)
  b[i] = rand() / ((double ) RAND_MAX);

 double *W = aligned_malloc((sizeof(double)) * (((M * 7) * N) * P), 32);
 for (int i = 0; i < (((M * 7) * N) * P); ++i)
  W[i] = (rand() / ((double ) RAND_MAX)) / 9.0;

 int repeat = 1;
 double runtime = 0.0;
 double wct_start;
 double wct_end;
 double cput_start;
 double cput_end;
```

```
  double *tmp;

  #ifdef LIKWID_PERFMON
  #pragma omp parallel
  {
   LIKWID_MARKER_START("Sweep");
  }
  #endif

  while (runtime < 0.5)
  {
   timing(&wct_start, &cput_start);
   for (int n = 0; n < repeat; ++n)
   {
    kernel_loop(a, b, W, M, N, P, block_factor);
    tmp = a;
    a = b;
    b = tmp;
   }

   timing(&wct_end, &cput_end);
   runtime = wct_end - wct_start;
   repeat *= 2;
  }


  #ifdef LIKWID_PERFMON
  #pragma omp parallel
  {
   LIKWID_MARKER_STOP("Sweep");
  }
  #endif

  repeat /= 2;
  printf("Performance in mlup/s: %lf\n", (((double ) repeat) * ((((double ) (M - 4)) * ((double ) (N - 4))) * ((double ) (P - 4)))) /
(runtime * 1000000.));
  printf("size: %d    time: %lf    iter: %d    mlup/s: %lf\n", (M * N * P), runtime, repeat, (((double ) repeat) * ((((double ) (M -
4)) * ((double ) (N - 4))) * ((double ) (P - 4)))) / (runtime * 1000000.));
  double total = 0.0;
  for (int k = 2; k < (M - 2); k++)
  {
   for (int j = 2; j < (N - 2); j++)
   {
    for (int i = 2; i < (P - 2); i++)
    {
     total = total + (a[(i + (j * P)) + (k * (P * N))] - b[(i + (j * P)) + (k * (P * N))]);
    }

   }

  }

  printf("diff(a-b): %lf\n", total);

  #ifdef LIKWID_PERFMON
  LIKWID_MARKER_CLOSE;
  #endif

}
```

and the following kernel.c:

```
#ifndef min
```

```
#define min( a, b ) ( ((a) < (b)) ? (a) : (b) )
#endif
void kernel_loop(double *a, double *b, double *W, int M, int N, int P, int block_factor)
{
  #pragma omp parallel
  for (int jb = 2; jb < (N - 2); jb += block_factor)
  {
    int jend = min(jb + block_factor, N - 2);
    #pragma omp for schedule(runtime)
    for (int k = 2; k < (M - 2); k++)
    {
      for (int j = jb; j < jend; j++)
      {
        for (int i = 2; i < (P - 2); i++)
        {
          b[(i + (j * P)) + (k * (P * N))] = ((((((W[((i + (j * P)) + (0 * (P * N))) + (k * ((P * N) * 7))] * a[(i + (j * P)) + (k * (P * N))]) +
(W[((i + (j * P)) + (1 * (P * N))) + (k * ((P * N) * 7))] * (a[((i - 1) + (j * P)) + (k * (P * N))] + a[((i + 1) + (j * P)) + (k * (P * N))]))) +
(W[((i + (j * P)) + (2 * (P * N))) + (k * ((P * N) * 7))] * (a[(i + (j * P)) + ((k - 1) * (P * N))] + a[(i + (j * P)) + ((k + 1) * (P * N))]))) +
(W[((i + (j * P)) + (3 * (P * N))) + (k * ((P * N) * 7))] * (a[(i + ((j - 1) * P)) + (k * (P * N))] + a[(i + ((j + 1) * P)) + (k * (P * N))]))) +
(W[((i + (j * P)) + (4 * (P * N))) + (k * ((P * N) * 7))] * (a[((i - 2) + (j * P)) + (k * (P * N))] + a[((i + 2) + (j * P)) + (k * (P * N))]))) +
(W[((i + (j * P)) + (5 * (P * N))) + (k * ((P * N) * 7))] * (a[(i + (j * P)) + ((k - 2) * (P * N))] + a[(i + (j * P)) + ((k + 2) * (P * N))]))) +
(W[((i + (j * P)) + (6 * (P * N))) + (k * ((P * N) * 7))] * (a[(i + ((j - 2) * P)) + (k * (P * N))] + a[(i + ((j + 2) * P)) + (k * (P * N))])));
        }

      }

    }

  }

}
```

In order to compile you then need some headers available in headers. An example of working compilation command is:
gcc -std=c99 -O3 -fopenmp -D_POSIX_C_SOURCE=200112L -Iheaders headers/timing.c stencil_compilable.c -o stencil.exe

Running:
./stencil.exe 200 200 250

causes the termination of the program:
Wrong number of arguments. Usage:
./stencil.exe size size size blocking

because in this example, a blocked version of the stecnil was used. So the correct command is:
./stencil.exe 200 200 250 64

This way the stencil was run with dimensions 200, 200 and 250 and a blocking factor of 64 on the middle loop.

If compiled enabling LIKWID_PERF:

gcc -std=c99 -O3 -fopenmp -march=native -fargument-noalias -pthread -D_POSIX_C_SOURCE=200112L -DLIKWID_PERFMON -Iheaders -I/apps/likwid/system/include/ -L/apps/likwid/system/lib headers/timing.c stencil_compilable.c -o stencil.exe -llikwid

can then be run this way:

likwid-perfctr -m -g L2CACHE -C S0:2 ./stencil.exe 250 250 250 64

and this is an example output (when run with the previous command line):

```
--------------------------------------------------------------------------------
CPU name:       Intel(R) Xeon(R) CPU         X5650  @ 2.67GHz
CPU type:       Intel Core Westmere processor
CPU clock:      2.67 GHz
--------------------------------------------------------------------------------
Performance in mlup/s: 90.453027
size: 15625000    time: 0.658328    iter: 4    mlup/s: 90.453027
diff(a-b): -288130.231583
--------------------------------------------------------------------------------
Region Sweep, Group 1: L2CACHE
+-------------------------+-------------------+
|    Region Info          |  Core 2           |
+-------------------------+-------------------+
| RDTSC Runtime [s] | 1.316398        |
|     call count          |     1             |
+-------------------------+-------------------+

+----------------------+---------+------------+
|        Event         | Counter |   Core 2 |
+------------------------------------ -+-----------+------------------+
|   INSTR_RETIRED_ANY          |  FIXC0   | 3684279000 |
| CPU_CLK_UNHALTED_CORE |  FIXC1   | 3521397000 |
|  CPU_CLK_UNHALTED_REF   |  FIXC2   | 3064407000 |
|  L2_RQSTS_REFERENCES      |   PMC0  |  386624000  |
|    L2_RQSTS_MISS              |   PMC1  |  183417800  |
+-------------------------------------+------------+------------------+

+------------------------------+---------------+
|       Metric                 |   Core 2      |
+------------------------------+---------------+
|  Runtime (RDTSC) [s]   |   1.3164     |
| Runtime unhalted [s]  |   1.3204     |
|      Clock [MHz]           | 3064.5259 |
|         CPI                  |   0.9558     |
|    L2 request rate        |   0.1049     |
|     L2 miss rate           |   0.0498     |
|     L2 miss ratio          |   0.4744     |
+------------------------------+---------------+
```