Codee Training Series

April 25-26, 2023

Shift Left Performance

Automated Code inspection for Performance

April 24th, 2023

First consideration about using Codee at NERSC

- First, remember to load the Codee module
  ```
  $ module load codee
  ```
- The flag --help lists all the options available in the Codee command-line tools
  ```
  $ pwreport --help
  $ pwloops --help
  $ pwdirectives --help
  ```
- You can run Codee command-line tools on the login nodes (no need to run them on the compute nodes)
- Build and run the example codes on the compute nodes using the batch scripts
  - Scripts tuned to use the appropriate reservations: *codee_day1*, *codee_day2*
- Remember to check the open catalog of rules for performance optimization:

https://www.codee.com/knowledge/

ATMUX

- Sparse linear algebra
- Multiplication of Transposed Sparse Matrix by Vector

$$y = A^T x = \begin{pmatrix} 0 & 40 & 0 & 0 & 0 \\ 0 & 55 & 56 & 9 & 34 \\ 18 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 57 \\ 0 & 21 & 0 & 29 & 0 \end{pmatrix}^T \cdot \begin{pmatrix} 83 \\ 86 \\ 77 \\ 15 \\ 93 \end{pmatrix} = \begin{pmatrix} 18 \times 77 \\ 40 \times 83 + 55 \times 86 + 21 \times 93 \\ 56 \times 86 \\ 9 \times 86 + 29 \times 93 \\ 34 \times 86 + 57 \times 15 \end{pmatrix} = \begin{pmatrix} 1386 \\ 10003 \\ 4816 \\ 3471 \\ 3779 \end{pmatrix}$$

- Using an sparse/compressed storage format: Compressed Row Storage (CRS)

$$A = \begin{array}{c} \begin{matrix} c0 & c1 & c2 & c3 & c4 \end{matrix} \\ \begin{pmatrix} 0 & 40 & 0 & 0 & 0 \\ 0 & 55 & 56 & 9 & 34 \\ 18 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 57 \\ 0 & 21 & 0 & 29 & 0 \end{pmatrix} \begin{matrix} r0 \\ r1 \\ r2 \\ r3 \\ r4 \end{matrix} \end{array}$$

val[9] = { 40, 55, 56, 9, 34, 18, 57, 21, 29 }

row_ptr[6] = { 0, 1, 5, 6, 7, 9 }

col_ind[9] = { 1, 1, 2, 3, 4, 0, 4, 1, 3 }

The source code of ATMUX

```
16  // Compute sparse matrix-vector multiplication
17  void atmux(double *val, double *x, double *y, int *col_ind, int *row_ptr, int n) {
18      for (int t = 0; t < n; t++)
19          y[t] = 0;
20
21      // y = A^T x
22      for (int i = 0; i < n; i++) {
23          for (int k = row_ptr[i]; k < row_ptr[i + 1]; k++) {
24              y[col_ind[k]] = y[col_ind[k]] + x[i] * val[k];
25          }
26      }
27  }
```

Produce Codee Checks Report for Multithreading and follow suggestions!

```
$ pwreport --checks --verbose atmux.c:atmux --include-tags multi -- -lm -fast -I lib/
Compiler flags: -lm -fast -I lib/

[C] target compiler: <none> (Compiler Agnostic Mode)

CHECKS REPORT

atmux.c:22:5 [PWR052]: consider applying multithreading parallelism to sparse reduction loop
  Suggestion: use pwdirectives to automatically optimize the code
  Documentation: https://www.codee.com/knowledge/pwr052
  AutoFix:
      * Using OpenMP 'for' with atomic protection (recommended):
        pwdirectives --multi omp-for --in-place atmux.c:22:5 -- -lm -fast -I lib/
      * Using OpenMP 'for' with explicit privatization:
        pwdirectives --multi omp-for --in-place --explicit-privatization y atmux.c:22:5 -- -lm -fast -I lib/
      * Using OpenMP 'taskwait':
        pwdirectives --multi omp-taskwait --in-place atmux.c:22:5 -- -lm -fast -I lib/
      * Using OpenMP 'taskloop':
        pwdirectives --multi omp-taskloop --in-place atmux.c:22:5 -- -lm -fast -I lib/

SUGGESTIONS

  More details on the defects, recommendations and more in the Knowledge Base:
        https://www.codee.com/knowledge/

1 file successfully analyzed and 0 failures in 28 ms
```

Generate code optimized for CPU with OpenMP directives (strategy based on Atomic Protection)

```
$ pwdirectives --multi omp-for -o atmux_atomic.c atmux.c:22:5 -- -lm -fast -I lib/
Compiler flags: -lm -fast -I lib/

[C] target compiler: <none> (Compiler Agnostic Mode)

Results for file 'atmux.c':
  Successfully parallelized loop at 'atmux.c:atmux:22:5' [using multi-threading]:
      [INFO] atmux.c:22:5 Parallel sparse reduction pattern identified for variable 'y' with associative, commutative operator '+'
      [INFO] atmux.c:22:5 Available parallelization strategies for variable 'y'
      [INFO] atmux.c:22:5   #1 OpenMP atomic access (* implemented)
      [INFO] atmux.c:22:5   #2 OpenMP explicit privatization
      [INFO] atmux.c:22:5 Loop parallelized with multithreading using OpenMP directive 'for'
      [INFO] atmux.c:22:5 Parallel region defined by OpenMP directive 'parallel'
      [INFO] atmux.c:22:5 Make sure there is no aliasing among variables: row_ptr, col_ind
      [INFO] atmux.c:22:5 Make sure there is no aliasing among variables: x, val, y
Successfully created atmux_atomic.c

Minimum software stack requirements: OpenMP version 3.0 with multithreading capabilities

$ cat -n atmux_atomic.c
...
    16    // Compute sparse matrix-vector multiplication
    17    void atmux(double *val, double *x, double *y, int *col_ind, int *row_ptr, int n) {
    18        for (int t = 0; t < n; t++)
    19            y[t] = 0;
    20
    21        // y = A^T x
    22        #pragma omp parallel default(none) shared(col_ind, n, row_ptr, val, x, y)
    23        {
    24        #pragma omp for schedule(auto)
    25        for (int i = 0; i < n; i++) {
    26            for (int k = row_ptr[i]; k < row_ptr[i + 1]; k++) {
    27                #pragma omp atomic update
    28                y[col_ind[k]] = y[col_ind[k]] + x[i] * val[k];
    29            }
    30        }
    31        } // end parallel
    32    }
...
```

Generate code optimized for CPU with OpenMP directives (strategy based on Explicit Privatization)

Codee generated a template code that requires the programmer to fill in additional information manually.

Lines to be fixed:

unsigned int y_length = 0 + n;

for (int i = 0; i < y_length; ++i) {

for(int i = 0; i < y_length; ++i) {

```
$ pwdirectives --multi omp-for --explicit-privatization y -o atmux_explicit.c atmux.c:22:5 -- -lm -fast -I lib/
Compiler flags: -lm -fast -I lib/

[C] target compiler: <none> (Compiler Agnostic Mode)

Results for file 'atmux.c':
  Successfully parallelized loop at 'atmux.c:atmux:22:5' [using multi-threading]:
      [INFO] atmux.c:22:5 Parallel sparse reduction pattern identified for variable 'y' with associative, commutative operator '+'
      [INFO] atmux.c:22:5 Available parallelization strategies for variable 'y'
      [INFO] atmux.c:22:5   #1 OpenMP explicit privatization (* implemented)
      [INFO] atmux.c:22:5   #2 OpenMP atomic access
      [INFO] atmux.c:22:5 Loop parallelized with multithreading using OpenMP directive 'for'
      [INFO] atmux.c:22:5 Please enter access range for variable 'y'
      [INFO] atmux.c:22:5 Critical section guarantees correctness of the reduction operation
      [INFO] atmux.cc:22:5 Parallel region defined by OpenMP directive 'parallel'
      [INFO] atmux.c:22:5 Make sure there is no aliasing among variables: row_ptr, col_ind
      [INFO] atmux.c:22:5 Make sure there is no aliasing among variables: x, val, y
Successfully created atmux_explicit.

Minimum software stack requirements: OpenMP version 3.0 with multithreading capabilities

$ cat -n atmux_explicit.c
...
    17      // Compute sparse matrix-vector multiplication
    18      void atmux(double *val, double *x, double *y, int *col_ind, int *row_ptr, int n) {
    19          for (int t = 0; t < n; t++)
    20              y[t] = 0;
    21
    22          // y = A^T x
    23          #pragma omp parallel default(none) shared(col_ind, n, row_ptr, val, x, y)
    24          {
    25          // preamble
    26          unsigned int y_length = /* y start */ + /* y length */;
    27          double *y_private = (double *) malloc(sizeof(double) * y_length);
    28          for (int i = /* y start */; i < y_length; ++i) {
    29            y_private[i] = 0;
    30          }
    31          // end preamble
    32          #pragma omp for schedule(auto)
    33          for (int i = 0; i < n; i++) {
    34              for (int k = row_ptr[i]; k < row_ptr[i + 1]; k++) {
    35                  y_private[col_ind[k]] = y_private[col_ind[k]] + x[i] * val[k];
    36              }
    37          }
    38          // postamble
    39          #pragma omp critical
    40          for(int i = /* y start */; i < y_length; ++i) {
    41            y[i] += y_private[i];
    42          }
    43          free(y_private);
    44          // end postamble
    45          } // end parallel
    46      }

...
```

Add missing information to the OpenMP code manually
(Codee generates template for Explicit Privatization)

```
$ cat -n atmux_explicit_fix.c
...
    18    void atmux(double *val, double *x, double *y, int *col_ind, int *row_ptr, int n) {
    19        for (int t = 0; t < n; t++)
    20            y[t] = 0;
    21
    22        // y = A^T x
    23        #pragma omp parallel default(none) shared(col_ind, n, row_ptr, val, x, y)
    24        {
    25        // preamble
    26        unsigned int y_length = 0 + n;
    27        double *y_private = (double *) malloc(sizeof(double) * y_length);
    28        for (int i = 0; i < y_length; ++i) {
    29          y_private[i] = 0;
    30        }
    31        // end preamble
    32        #pragma omp for schedule(auto)
    33        for (int i = 0; i < n; i++) {
    34            for (int k = row_ptr[i]; k < row_ptr[i + 1]; k++) {
    35                y_private[col_ind[k]] = y_private[col_ind[k]] + x[i] * val[k];
    36            }
    37        }
    38        // postamble
    39        #pragma omp critical
    40        for(int i = 0; i < y_length; ++i) {
    41          y[i] += y_private[i];
    42        }
    43        free(y_private);
    44        // end postamble
    45        } // end parallel
    46    }
...
```

Benchmarking on Perlmutter CPU @NERSC using GNU toolchain

Launch script “launch_CPU.sh”

```
#!/bin/bash
#SBATCH -A ntrain8
#SBATCH --reservation=codee_day1_cpu
#SBATCH -C cpu
#SBATCH -J Codee_ATMUX
#SBATCH -q regular
#SBATCH -t 0:20:00
#SBATCH -N 1
#SBATCH --ntasks=1          #Indicate the maximum number of processes
#SBATCH --ntasks-per-node=1 #Indicate how many tasks per node you want to run
#SBATCH --cpus-per-task=32  #Indicate how many CPU cores per task you need

export SLURM_CPU_BIND="cores"
export OMP_NUM_THREADS=32

srun -N 1 -n 1 ATMUX_CPU.sh
```

ATMUX execution script “ATMUX_CPU.sh”

```
#!/bin/bash
...
module load PrgEnv-gnu
rm -f atmux atmux_atomic atmux_explicit

gcc -Ofast atmux.c lib/Matrix2D.c lib/Vector.c lib/CRSMatrix.c -o atmux -I lib
./atmux 17000

gcc -Ofast -fopenmp atmux_atomic.c lib/Matrix2D.c lib/Vector.c lib/CRSMatrix.c -o
atmux_atomic -I lib
./atmux_atomic 17000

gcc -Ofast -fopenmp atmux_explicit.c lib/Matrix2D.c lib/Vector.c lib/CRSMatrix.c
-o atmux_explicit -I lib
./atmux_explicit 17000
```

```
$ gcc -Ofast atmux.c lib/Matrix2D.c lib/Vector.c lib/CRSMatrix.c -o atmux -I lib
$ ./atmux 17000
- Input parameters
size     = 17000
- Executing test...
time (s)= 0.762878
size     = 17000
sparsity= 0.66
chksum   = 244110871193
iters    = 10

$ gcc -Ofast -fopenmp atmux_atomic.c lib/Matrix2D.c lib/Vector.c lib/CRSMatrix.c -o atmux_atomic -I lib
$ ./atmux_atomic 17000
- Input parameters
size     = 17000
- Executing test...
time (s)= 0.779836
size     = 17000
sparsity= 0.66
chksum   = 244110871193
iters    = 10

$ gcc -Ofast -fopenmp atmux_explicit.c lib/Matrix2D.c lib/Vector.c lib/CRSMatrix.c -o atmux_explicit -I lib
$ ./atmux_explicit 17000
- Input parameters
size     = 17000
- Executing test...
time (s)= 0.284465
size     = 17000
sparsity= 0.66
chksum   = 244110871193
iters    = 10
```

Atomic yields speedup 0.98x

Explicit privatization yields speedup 2.71x

Produce Codee Checks Report for Offloading and follow suggestions!

```
$ pwreport --checks --verbose atmux.c:atmux --include-tags gpu -- -lm -fast -I lib/
Compiler flags: -lm -fast -I lib/

[C] target compiler: <none> (Compiler Agnostic Mode)

CHECKS REPORT

atmux.c:22:5 [PWR035]: avoid non-consecutive array access for variables 'row_ptr' and 'x' to improve performance
  Non-consecutive array access:
    23:         for (int k = row_ptr[i]; k < row_ptr[i + 1]; k++) {
    24:             y[col_ind[k]] = y[col_ind[k]] + x[i] * val[k];
  Suggestion: consider using techniques like loop fusion, loop interchange, loop tiling or changing the data layout to avoid non-sequential access to variables 'row_ptr' and 'x'.
  Documentation: https://www.codee.com/knowledge/pwr035

atmux.c:22:5 [PWR036]: avoid indirect array access for variable 'y' to improve performance
  Non-consecutive array access:
    24:             y[col_ind[k]] = y[col_ind[k]] + x[i] * val[k];
  Suggestion: consider using techniques like loop fusion, loop interchange, loop tiling or changing the data layout to avoid non-sequential access to variable 'y'.
  Documentation: https://www.codee.com/knowledge/pwr036

atmux.c:22:5 [PWR057]: consider applying offloading parallelism to sparse reduction loop
  Suggestion: use pwdirectives to automatically optimize the code
  Documentation: https://www.codee.com/knowledge/pwr057
  AutoFix (choose one option):
    * Using OpenMP (recommended):
        pwdirectives --offload omp-teams --in-place atmux.c:22:5 -- -lm -fast -I lib/
    * Using OpenAcc:
        pwdirectives --offload acc --in-place atmux.c:22:5 -- -lm -fast -I lib/

atmux.c:23:9 [PWR036]: avoid indirect array access for variable 'y' to improve performance
  Non-consecutive array access:
    24:             y[col_ind[k]] = y[col_ind[k]] + x[i] * val[k];
  Suggestion: consider using techniques like loop fusion, loop interchange, loop tiling or changing the data layout to avoid non-sequential access to variable 'y'.
  Documentation: https://www.codee.com/knowledge/pwr036

SUGGESTIONS

  More details on the defects, recommendations and more in the Knowledge Base:
        https://www.codee.com/knowledge/

1 file successfully analyzed and 0 failures in 31 ms
```

Generate GPU code using OpenMP offload directives

Codee generated a template code that requires the programmer to fill in additional information manually.

Lines to be fixed:

atmux (..., int n, long long nnz) {

map(to: col_ind[0:nnz], n, row_ptr[0:n+1], val[0:nnz], x[0:n])

map(tofrom: y[0:n]) schedule(static)

```
$ pwdirectives --offload omp-teams -o atmux_ompGPU.c atmux.c:22:5 -- -lm -fast -I lib/
Compiler flags: -lm -fast -I lib/

[C] target compiler: <none> (Compiler Agnostic Mode)

Results for file 'atmux.c':
  Successfully parallelized loop at 'atmux.c:atmux:22:5' [using offloading]:
      [INFO] atmux.c:22:5 Parallel sparse reduction pattern identified for variable 'y' with associative, commutative operator
'+'
      [INFO] atmux.c:22:5 Available parallelization strategies for variable 'y'
      [INFO] atmux.c:22:5   #1 OpenMP atomic access (* implemented)
      [INFO] atmux.c:22:5   #2 OpenMP explicit privatization
      [INFO] atmux.c:22:5 Complete access range for variables: 'col_ind', 'val', 'y'
      [INFO] atmux.c:22:5 Loop parallelized with teams using OpenMP directive 'target teams distribute parallel for'
Successfully created atmux_ompGPU.c

Minimum software stack requirements: OpenMP version 4.0 with offloading capabilities


$ cat -n atmux_ompGPU.c
...
    16    // Compute sparse matrix-vector multiplication
    17    void atmux(double *val, double *x, double *y, int *col_ind, int *row_ptr, int n) {
    18        for (int t = 0; t < n; t++)
    19            y[t] = 0;
    20
    21        // y = A^T x
    22        #pragma omp target teams distribute parallel for shared(col_ind, n, row_ptr, val, x) map(to: col_ind[:], n,
row_ptr[0:n+1], val[:], x[0:n]) map(tofrom: y[:]) schedule(static)
    23        for (int i = 0; i < n; i++) {
    24            for (int k = row_ptr[i]; k < row_ptr[i + 1]; k++) {
    25                #pragma omp atomic update
    26                y[col_ind[k]] = y[col_ind[k]] + x[i] * val[k];
    27            }
    28        }
    29    }
...
    65            atmux(CRSMatrix_getData(in_sparseMat), Vector_getData(in_vec), Vector_getData(out_vec),
    66                  CRSMatrix_colRef(in_sparseMat), CRSMatrix_rowRef(in_sparseMat), param_n);
...
```

Generate GPU code using OpenACC offload directives

Codee generated a template code that requires the programmer to fill in additional information manually.

Lines to be fixed:

atmux (..., int n, long long nnz) {

copyin (col_ind[0:nnz], ...)

copyin (val[0:nnz], ...)

copy(y[0:n])

```
$ pwdirectives --offload acc -o atmux_acc.c atmux.c:22:5 -- -lm -fast -I lib/
Compiler flags: -lm -fast -I lib/

[C] target compiler: <none> (Compiler Agnostic Mode)

Results for file 'atmux.c':
  Successfully parallelized loop at 'atmux.c:atmux:22:5' [using offloading without teams]:
      [INFO] atmux.c:22:5 Parallel sparse reduction pattern identified for variable 'y' with associative, commutative operator '+'
      [INFO] atmux.c:22:5 Available parallelization strategies for variable 'y'
      [INFO] atmux.c:22:5   #1 OpenACC atomic access (* implemented)
      [INFO] atmux.c:22:5 Parallel region defined by OpenACC directive 'parallel'
      [INFO] atmux.c:22:5 Loop parallelized with OpenACC directive 'loop'
      [INFO] atmux.c:22:5 Complete access range for variables: 'col_ind', 'val', 'y'
      [INFO] atmux.c:22:5 Data region for host-device data transfers defined by OpenACC directive 'data'
      [INFO] atmux.c:22:5 Make sure there is no aliasing among variables: row_ptr, col_ind
      [INFO] atmux.c:22:5 Make sure there is no aliasing among variables: x, val, y
Successfully created atmux_acc.c

Minimum software stack requirements: OpenACC version 2.0 with offloading capabilities

$ cat -n atmux_acc.c
...
    16    // Compute sparse matrix-vector multiplication
    17    void atmux(double *val, double *x, double *y, int *col_ind, int *row_ptr, int n) {
    18        for (int t = 0; t < n; t++)
    19            y[t] = 0;
    20
    21        // y = A^T x
    22        #pragma acc data copyin(col_ind[:], n, row_ptr[0:n+1], val[:], x[0:n]) copy(y[:])
    23        #pragma acc parallel
    24        #pragma acc loop
    25        for (int i = 0; i < n; i++) {
    26            for (int k = row_ptr[i]; k < row_ptr[i + 1]; k++) {
    27                #pragma acc atomic update
    28                y[col_ind[k]] = y[col_ind[k]] + x[i] * val[k];
    29            }
    30        }
    31    }
...
    68            atmux(CRSMatrix_getData(in_sparseMat), Vector_getData(in_vec), Vector_getData(out_vec),
    69                  CRSMatrix_colRef(in_sparseMat), CRSMatrix_rowRef(in_sparseMat), param_n);

...
```

Add missing information to OpenMP/ OpenACC code manually

```
$ cat -n atmux_ompGPU_fix.c
...
    16      // Compute sparse matrix-vector multiplication
    17      void atmux(double *val, double *x, double *y, int *col_ind, int *row_ptr, int n, long long nnz) {
    18          for (int t = 0; t < n; t++)
    19              y[t] = 0;
    20          // y = A^T x
    21          #pragma omp target teams distribute parallel for shared(col_ind, n, row_ptr, val, x) map(to: col_ind[0:nnz], n, row_ptr[0:n+1], val[0:nnz], x[0:n]) map(tofrom: y[0:n]) schedule(static)
    22          for (int i = 0; i < n; i++) {
    23              for (int k = row_ptr[i]; k < row_ptr[i + 1]; k++) {
    24                  #pragma omp atomic update
    25                  y[col_ind[k]] = y[col_ind[k]] + x[i] * val[k];
    26              }
    27          }
    28      }
...
    65              atmux(CRSMatrix_getData(in_sparseMat), Vector_getData(in_vec), Vector_getData(out_vec),
    66                    CRSMatrix_colRef(in_sparseMat), CRSMatrix_rowRef(in_sparseMat), param_n, CRSMatrix_getSize(in_sparseMat));
...
```

```
$ cat -n atmux_acc_fix.c
...
    16      // Compute sparse matrix-vector multiplication
    17      void atmux(double *val, double *x, double *y, int *col_ind, int *row_ptr, int n, long long nnz) {
    18          for (int t = 0; t < n; t++)
    19              y[t] = 0;
    20
    21          // y = A^T x
    22          #pragma acc data copyin(col_ind[0:nnz], n, row_ptr[0:n+1], val[0:nnz], x[0:n]) copy(y[0:n])
    23          #pragma acc parallel
    24          #pragma acc loop
    25          for (int i = 0; i < n; i++) {
    26              for (int k = row_ptr[i]; k < row_ptr[i + 1]; k++) {
    27                  #pragma acc atomic update
    28                  y[col_ind[k]] = y[col_ind[k]] + x[i] * val[k];
    29              }
    30          }
    31      }
...
    68              atmux(CRSMatrix_getData(in_sparseMat), Vector_getData(in_vec), Vector_getData(out_vec),
    69                    CRSMatrix_colRef(in_sparseMat), CRSMatrix_rowRef(in_sparseMat), param_n, CRSMatrix_getSize(in_sparseMat));
...
```

Benchmarking on Perlmutter GPU @NERSC using Nvidia toolchain

Launch script "launch_GPU.sh"

```
#!/bin/bash
#SBATCH -A ntrain8
#SBATCH --reservation=codee_day1_gpu
#SBATCH -C gpu
#SBATCH -J Codee_Matmul_C
#SBATCH -q regular
#SBATCH -t 0:10:00
#SBATCH -N 1
#SBATCH --ntasks-per-node=1
#SBATCH -c 128
#SBATCH --gpus-per-task=1

export SLURM_CPU_BIND="cores"
srun ATMUX_GPU.sh
```

ATMUX execution script "ATMUX_GPU.sh"

```
#!/bin/bash
...
module load PrgEnv-gnu
rm -f *.o atmux atmux_ompGPU atmux_acc

nvc -fast atmux.c lib/Matrix2D.c lib/Vector.c lib/CRSMatrix.c -o atmux -I lib
./atmux 17000

nvc -fast -mp -target=gpu -Minfo=mp atmux_ompGPU.c lib/Matrix2D.c lib/Vector.c
lib/CRSMatrix.c -o atmux_ompGPU -I lib
./atmux_ompGPU 17000

nvc -fast -acc -target=gpu -Minfo=acc atmux_acc.c lib/Matrix2D.c lib/Vector.c
lib/CRSMatrix.c -o atmux_acc -I lib
./atmux_acc 17000
```

```
$ nvc -fast atmux.c lib/Matrix2D.c lib/Vector.c lib/CRSMatrix.c -o atmux -I lib
$ ./atmux 17000
- Input parameters
size     = 17000
- Executing test...
time (s)= 0.791512
size     = 17000
sparsity= 0.66
chksum   = 244110871193
iters    = 10

$ nvc -fast -mp -target=gpu -Minfo=mp atmux_ompGPU.c lib/Matrix2D.c lib/Vector.c lib/CRSMatrix.c -o atmux_ompGPU -I lib
$ ./atmux_ompGPU 17000
- Input parameters
size     = 17000
- Executing test...
time (s)= 1.438825
size     = 17000
sparsity= 0.66
chksum   = 244110871193
iters    = 10

$ nvc -fast -acc -target=gpu -Minfo=acc atmux_acc.c lib/Matrix2D.c lib/Vector.c lib/CRSMatrix.c -o atmux_acc -I lib
$ ./atmux_acc 17000
- Input parameters
size     = 17000
- Executing test...
time (s)= 1.377705
size     = 17000
sparsity= 0.66
chksum   = 244110871193
iters    = 10
```

OpenMP offload yields speedup 0.55x

OpenACC offload yields speedup 0.57x

ATMUX code runs correctly on the GPU @perlmutter using OpenACC/OpenMP offload, but runs slower because further optimizations are required. Challenge: in ATMUX increasing the problem size also increases the memory requirements, and thus the volume of data to be transferred CPU-GPU!

codee

www.codee.com

info@codee.com

Subscribe: codee.com/newsletter/

USA - Spain

codee_com

company/codee-com/