The Stencil Method

Mersimoski Kjanija

KjMercy/HPC_Assignment

Parallelization of Heat Equation

- Stencil computation for 2D heat diffusion
- MPI --> domain decomposition
- Further optimization with OpenMP

Serial

- 5 Point Stencil
- **Update rule**: weighted combination of self and neighbours
- Heat sources periodically inject energy
- **Naive solution**: double loop over all points N times

MPI

- Domain decomposition
  - Subdomain per task
  - Communication for data exchange
- Communication-computation **overlap**
- Non-blocking calls for neighbouring communication

Overlap

```
inject_energy(...);

//...

fill_buffers(&planes[current], buffers, planes[current].size, neighbours);
perform_halo_comms(buffers, neighbours, &myCOMM_WORLD, reqs, planes[current].size);


update_inner_plane(&planes[current], &planes[!current]);

//...

MPI_Waitall(8, reqs, statuses);

copy_halo_data(&planes[current], buffers, planes[current].size, neighbours);
update_border_plane(periodic, N, &planes[current], &planes[!current]);
```

Non Blocking Communications

```
void perform_halo_comms(buffers_t *buffers, int *neighbours, MPI_Comm *comm, MPI_Request *reqs, vec2_t size)
{
  for (uint i = 0; i < 8; i++)
  {
    reqs[i] = MPI_REQUEST_NULL;
  }

  if (neighbours[NORTH] != MPI_PROC_NULL)
  {
    MPI_Irecv(buffers[RECV][NORTH], size[_x_], MPI_DOUBLE, neighbours[NORTH], 0, *comm, &reqs[0]);
    MPI_Isend(buffers[SEND][NORTH], size[_x_], MPI_DOUBLE, neighbours[NORTH], 1, *comm, &reqs[1]);
  }

  if (neighbours[SOUTH] != MPI_PROC_NULL)
  {
    MPI_Irecv(...);
    MPI_Isend(...);
  }

  if (neighbours[EAST] != MPI_PROC_NULL)
  {//...
  }
  if (neighbours[WEST] != MPI_PROC_NULL)
  {//...
  }
}
```

OpenMP

```
#pragma omp parallel for collapse(2) schedule(static)
  for (uint j = 2; j <= ysize - 1; j++)
    for (uint i = 2; i <= xsize - 1; i++)
      new[IDX(i, j)] = stencil_computation(old, fxsize, i, j);
```

```
#pragma omp parallel for schedule(static)
  for (uint j = 1; j <= ysize; j++)
  {
    new[IDX(1, j)] = stencil_computation(old, fxsize, 1, j);          // left border
    new[IDX(xsize, j)] = stencil_computation(old, fxsize, xsize, j); // right border
  }

#pragma omp parallel for schedule(static)
  for (uint i = 1; i <= xsize; i++)
  {
    new[IDX(i, 1)] = stencil_computation(old, fxsize, i, 1);          // top border
    new[IDX(i, ysize)] = stencil_computation(old, fxsize, i, ysize); // bottom border
  }
```

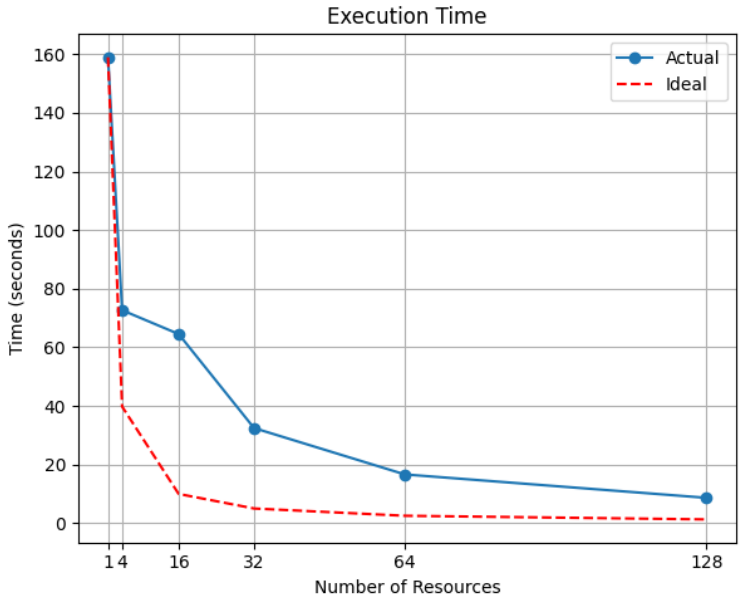
OpenMP

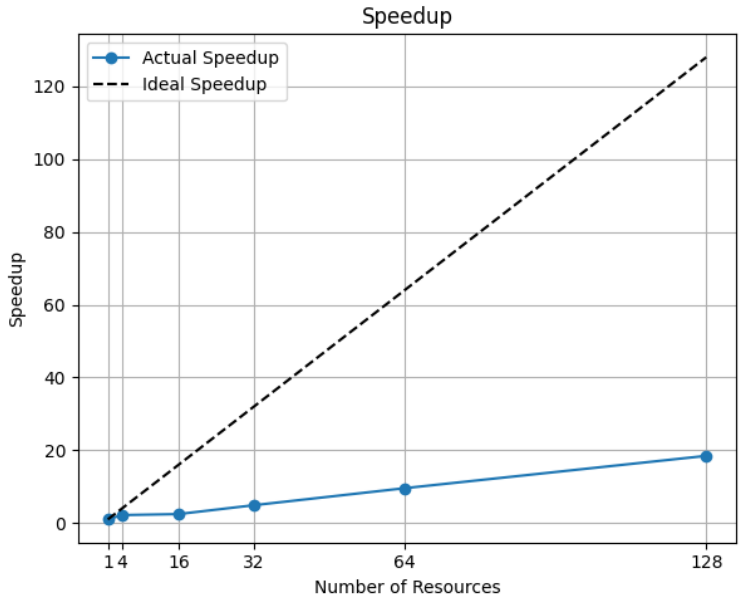
```
#pragma omp parallel for reduction(+ : totenergy) collapse(2) schedule(static)
    for (int j = 1; j <= ysize; j++)
        for (int i = 1; i <= xsize; i++)
            totenergy += data[IDX(i, j)];
```

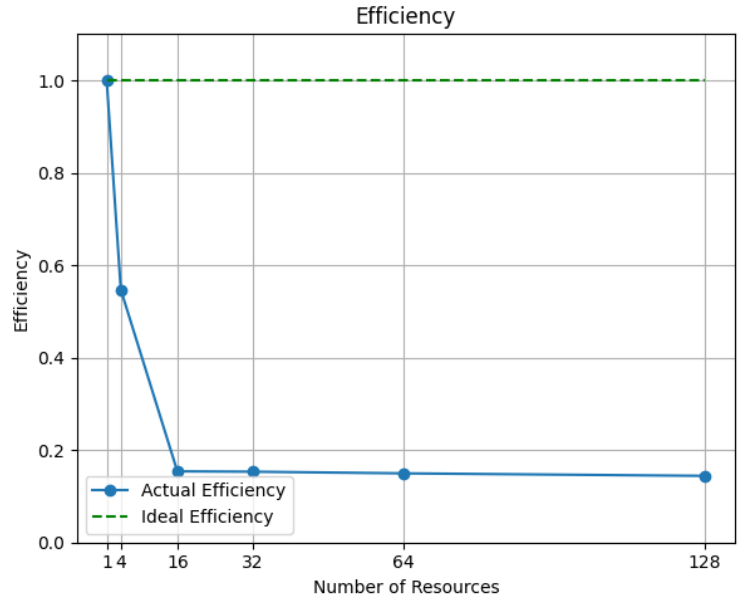
Vectorization

```
#pragma omp parallel for
  for (uint j = 2; j <= ysize - 1; j++)
  {
    uint i = 2;
    for (; i <= simd_end; i += simd_width) // SIMD loop
    {
      __m256d center = _mm256_loadu_pd(&old[IDX(i, j)]);
      __m256d left = _mm256_loadu_pd(&old[IDX(i - 1, j)]);
      __m256d right = _mm256_loadu_pd(&old[IDX(i + 1, j)]);
      __m256d up = _mm256_loadu_pd(&old[IDX(i, j - 1)]);
      __m256d down = _mm256_loadu_pd(&old[IDX(i, j + 1)]);

      __m256d res = _mm256_add_pd(_mm256_mul_pd(center, _mm256_set1_pd(0.5)),
                                  _mm256_mul_pd(_mm256_add_pd(_mm256_add_pd(left, right),
                                                              _mm256_add_pd(up, down)),
                                                _mm256_set1_pd(0.125)));
      _mm256_storeu_pd(&new[IDX(i, j)], res);
    }

    for (; i <= xsize - 1; i++)
      new[IDX(i, j)] = stencil_computation(old, fxsize, i, j);
  }
```

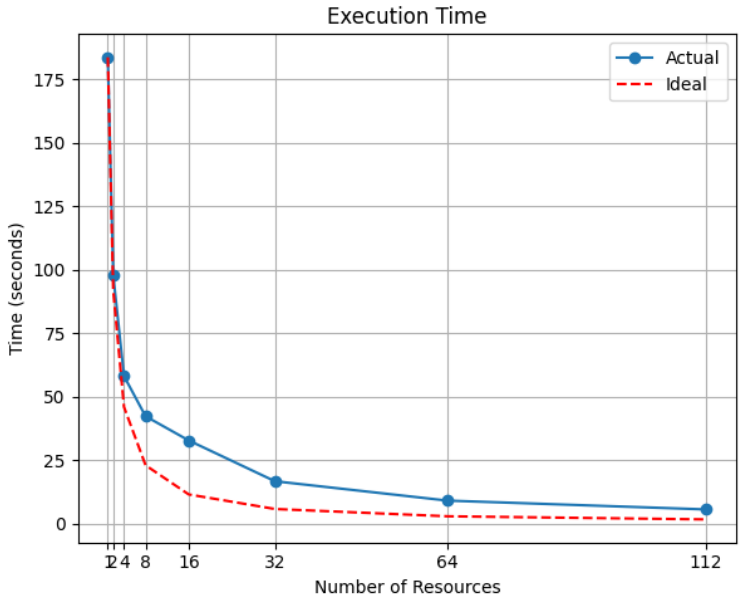
Threads Scaling - Orfeo

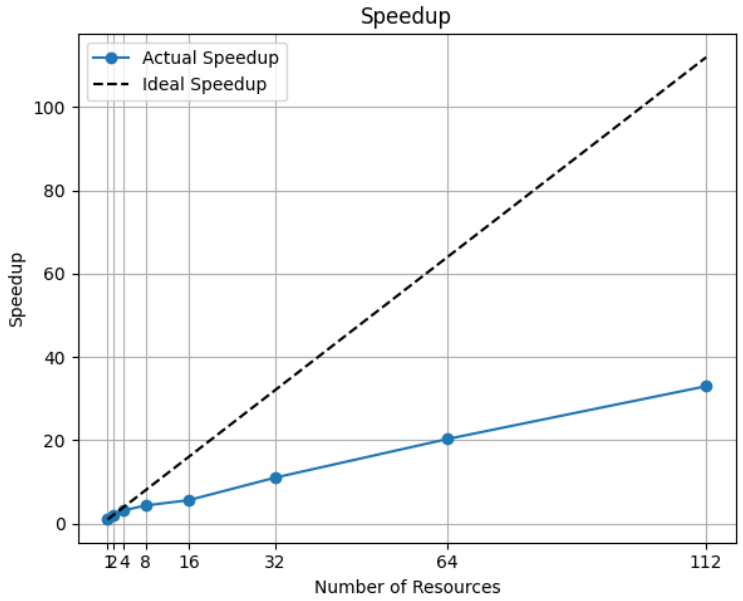


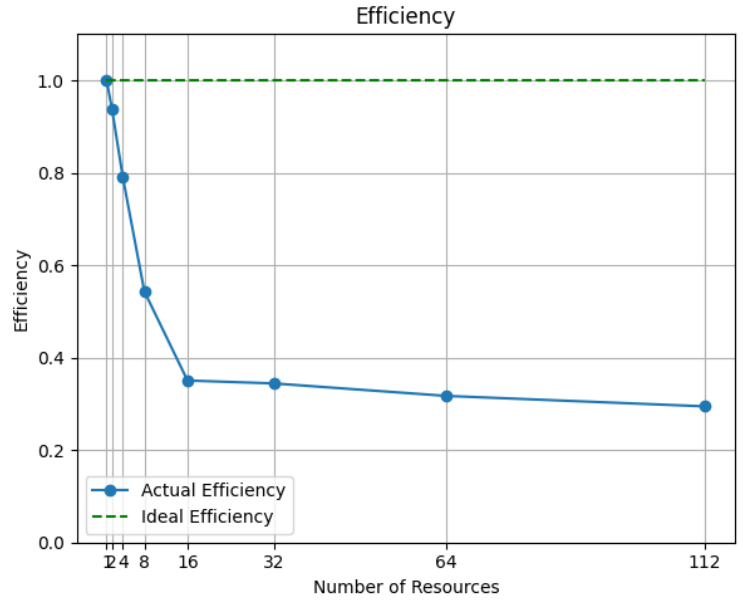




Threads Scaling - Leonardo



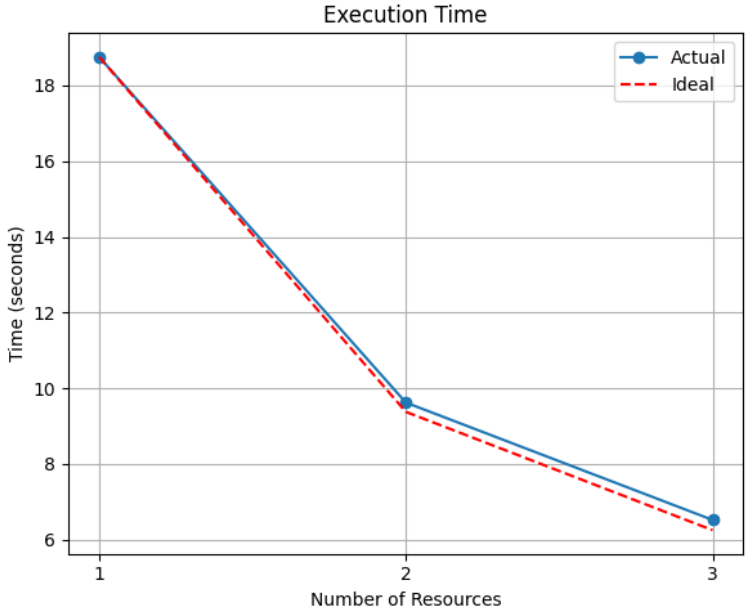


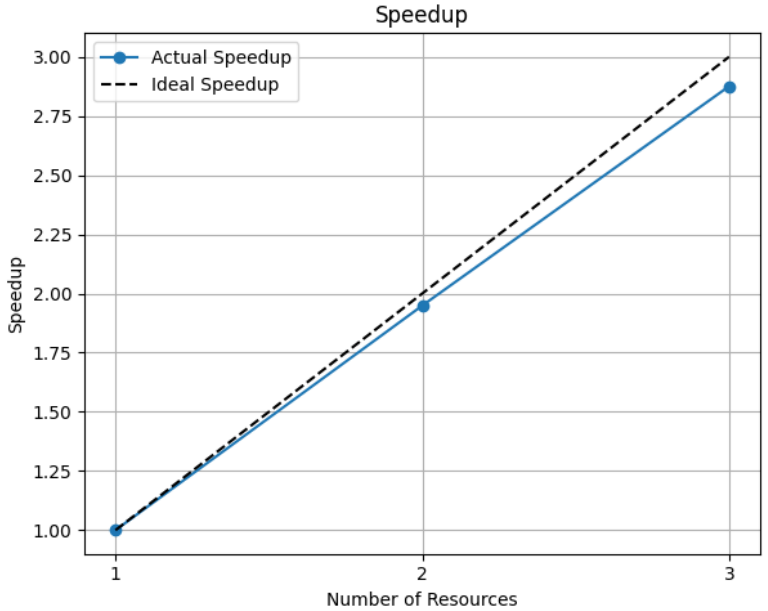


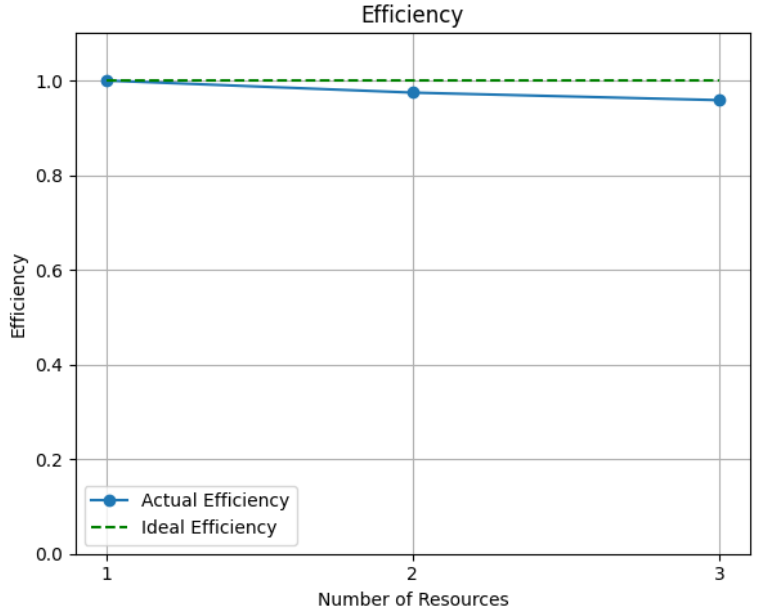
Strong Scaling

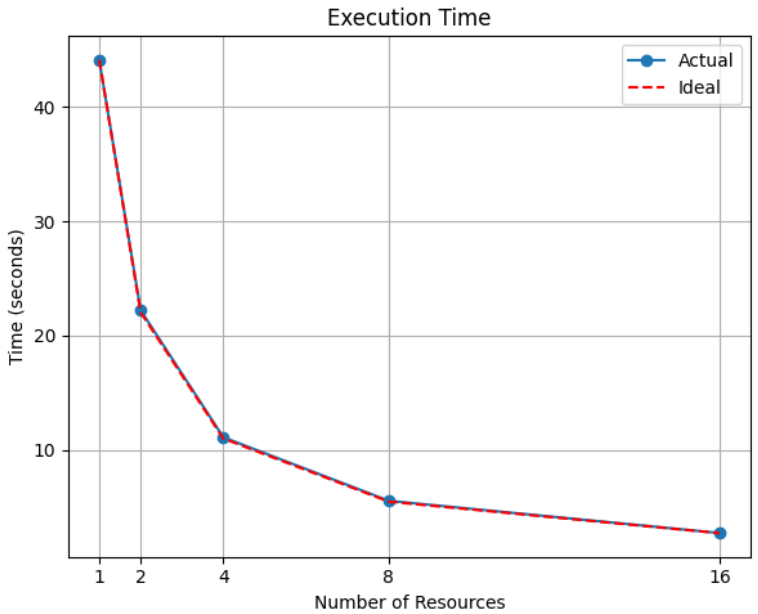
- Nodes: 1, 2, 4, 8, 16
- Tasks per node: 8
- CPUs per task: 14

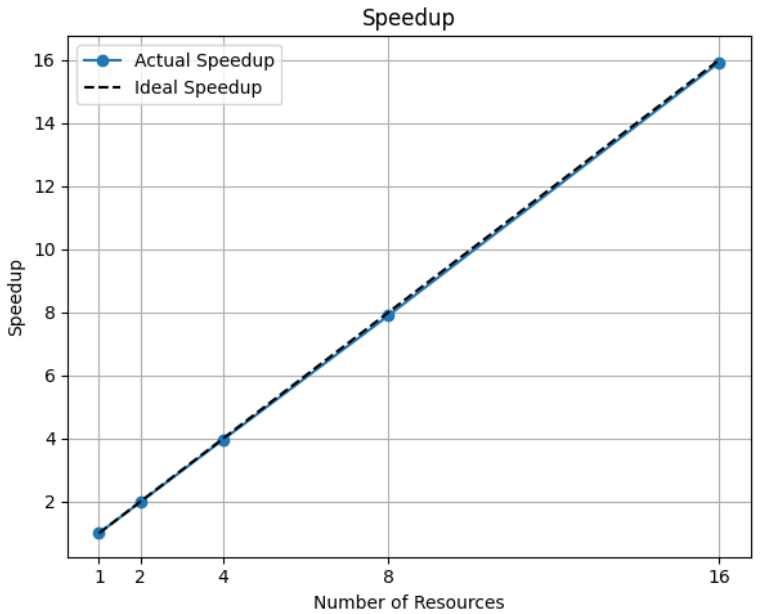
DGCP: 2x56 cores per node.

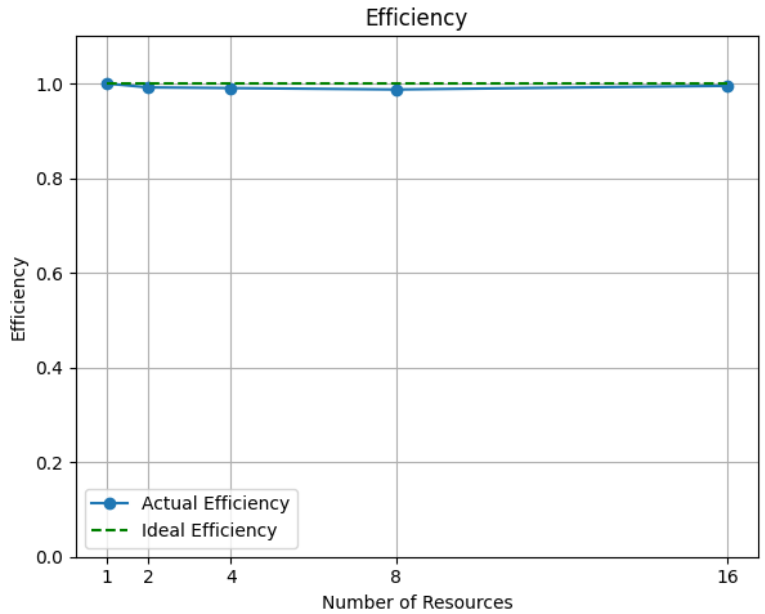

Execution Time
Actual
Ideal
Time (seconds)
Number of Resources


Speedup
Actual Speedup
Ideal Speedup
Speedup
Number of Resources

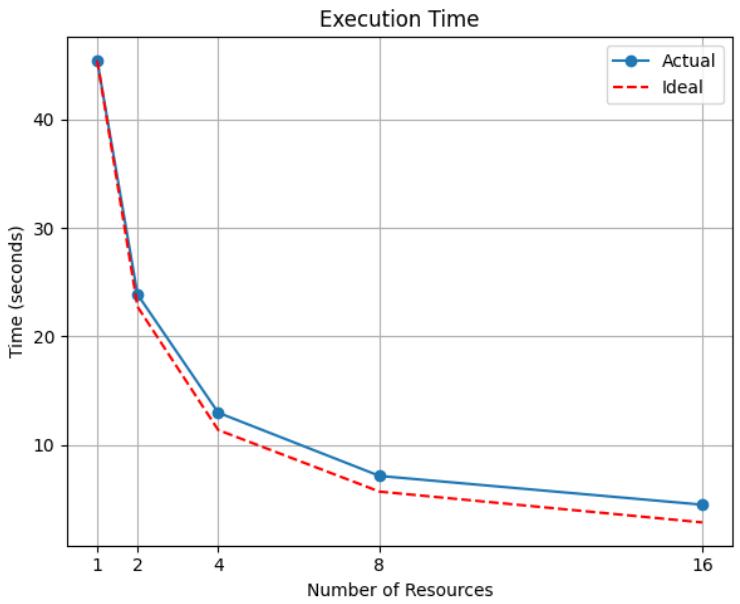

Efficiency
Actual Efficiency
Ideal Efficiency
Efficiency
Number of Resources

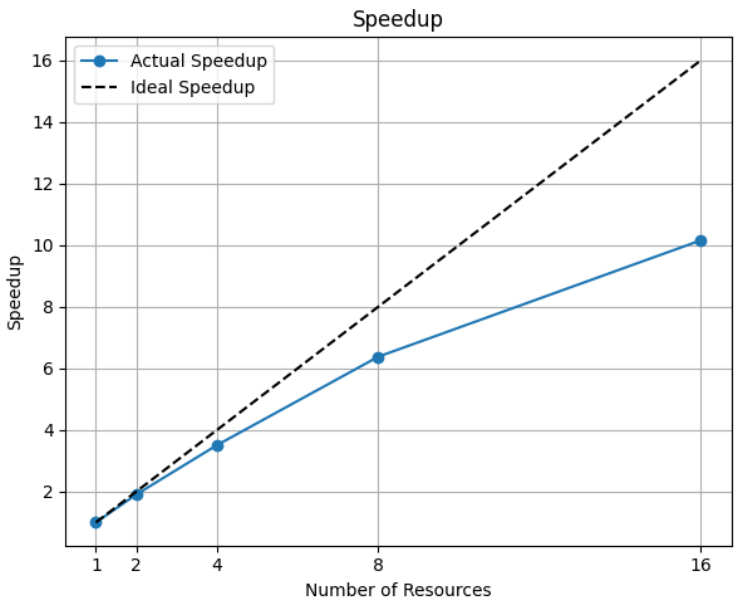

Execution Time
Actual
Ideal
Time (seconds)
Number of Resources

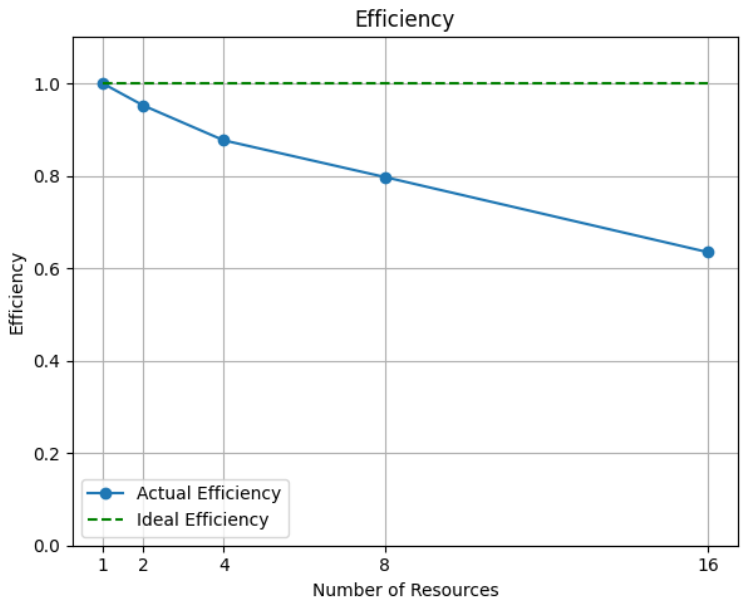

Speedup
Actual Speedup
Ideal Speedup
Speedup
Number of Resources


Efficiency
Actual Efficiency
Ideal Efficiency
Efficiency
Number of Resources

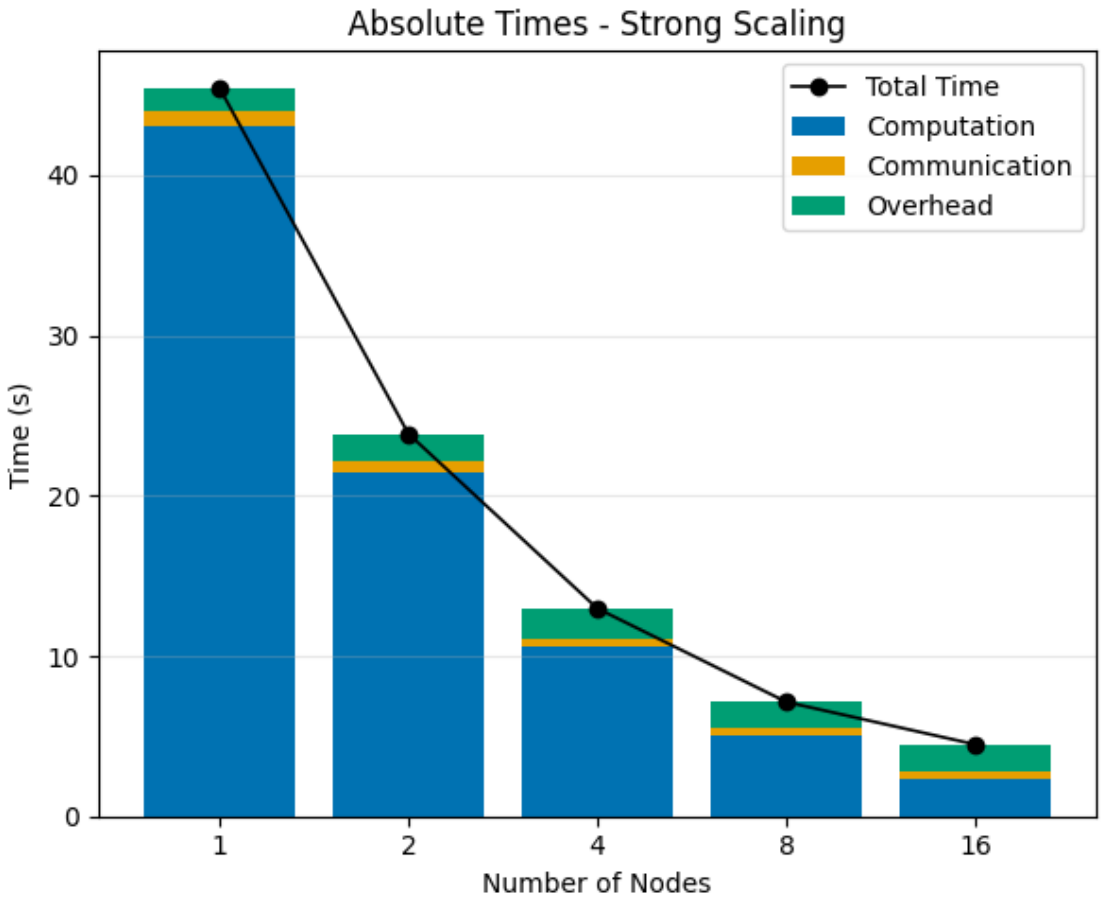
Strong Scaling - Total time

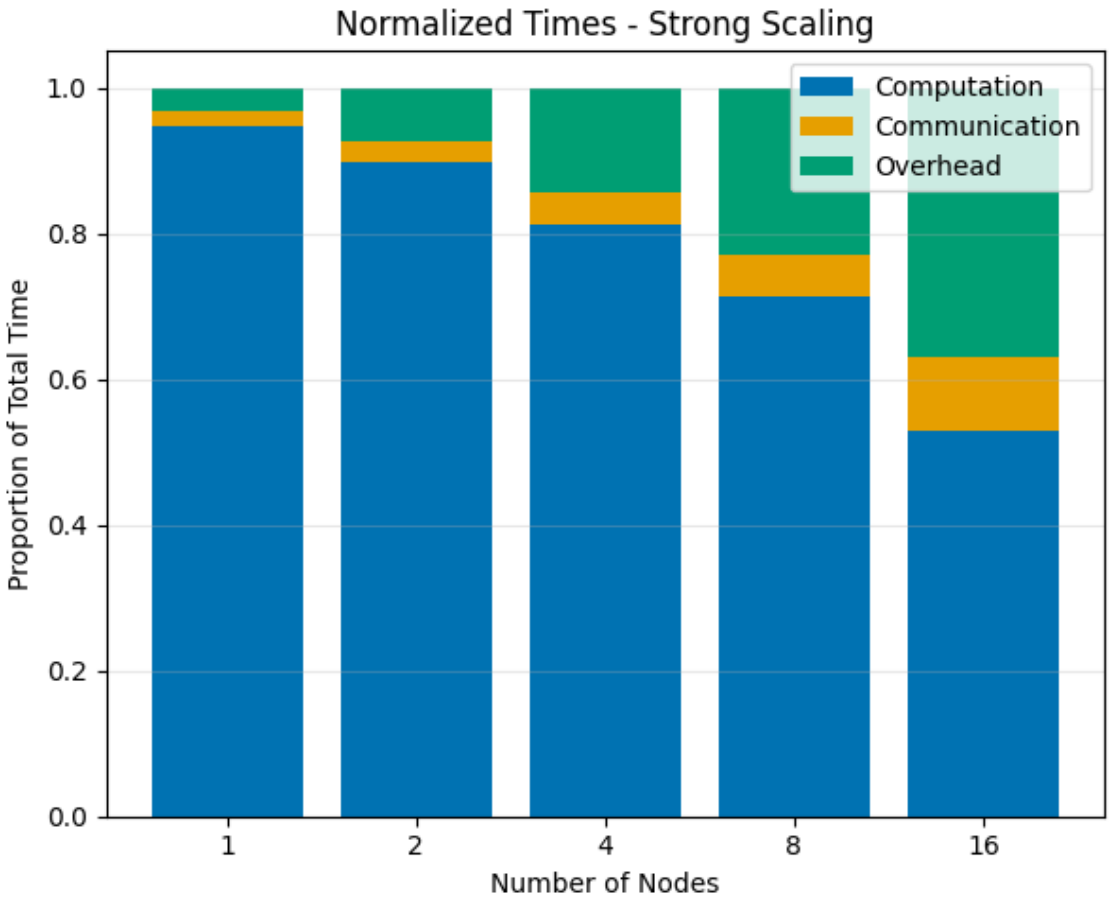






Time spent doing…





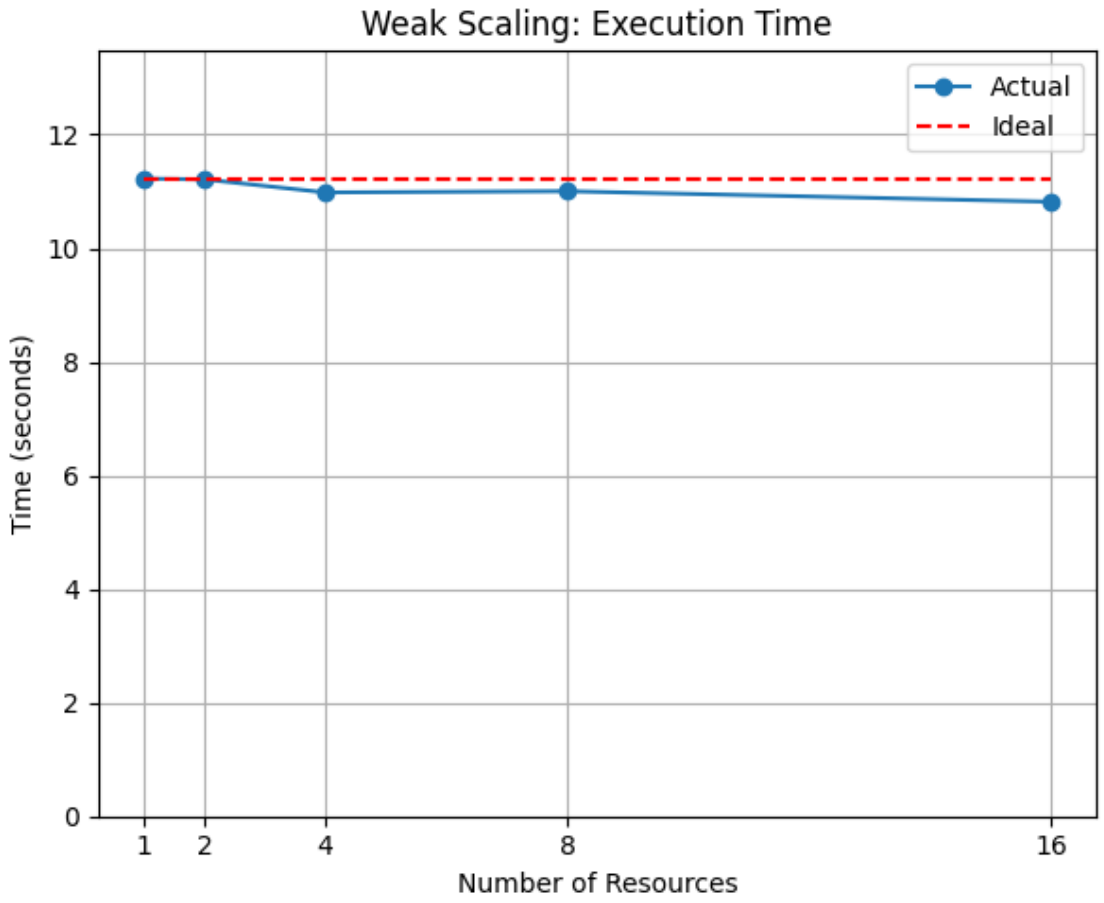
Weak Scaling

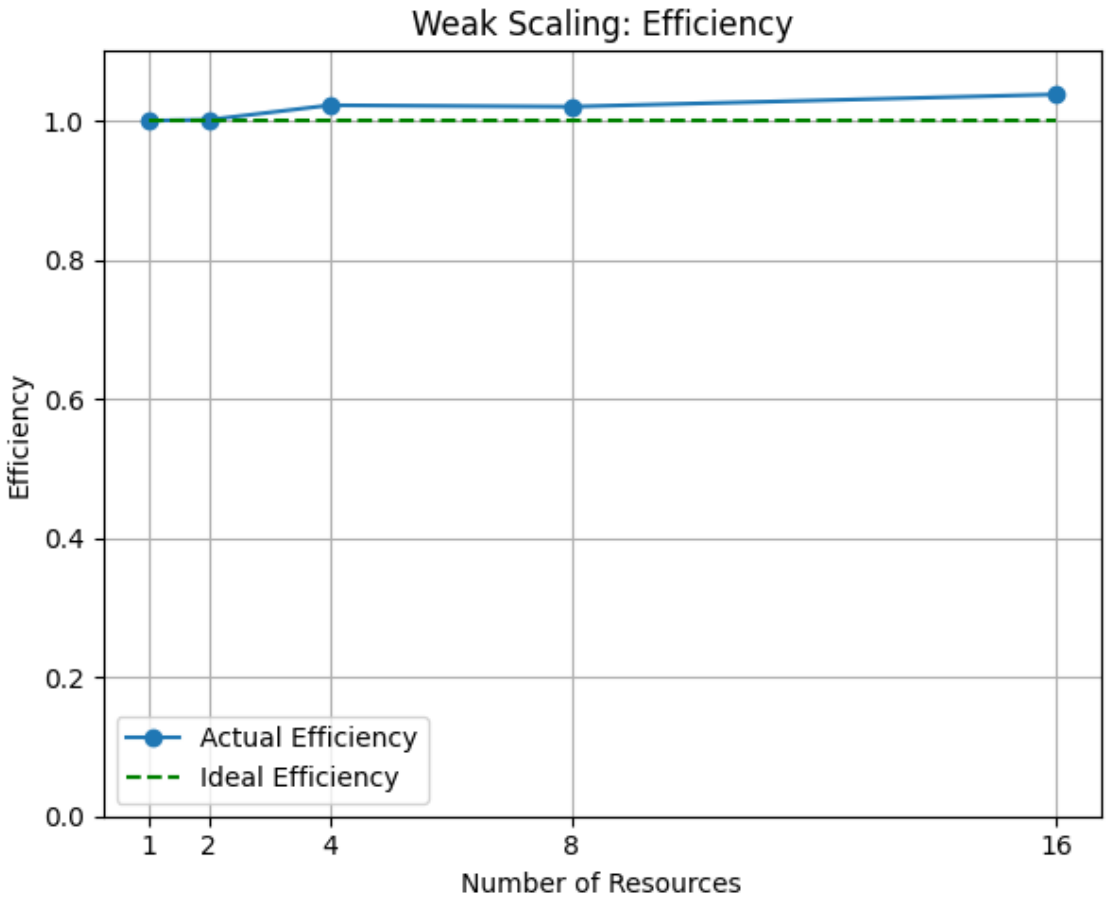
- Nodes: 1, 2, 4, 8, 16
- Tasks per node: 8
- CPUs per task: 14

$$x = y = x_0 \times \sqrt{\frac{\mathrm{nt}}{\mathrm{nt}_0}}$$

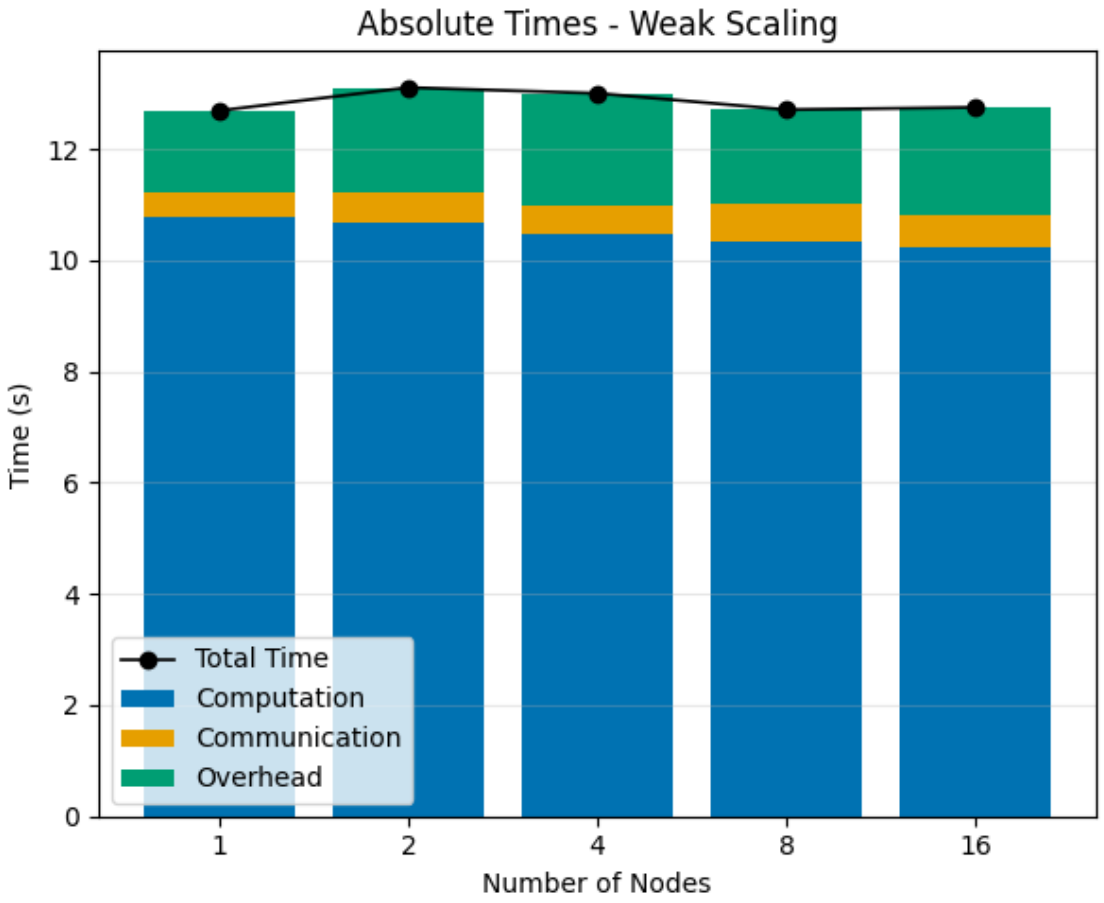
Where $\mathrm{nt}_0 = 8$ and $x_0 = y_0 = 15000$

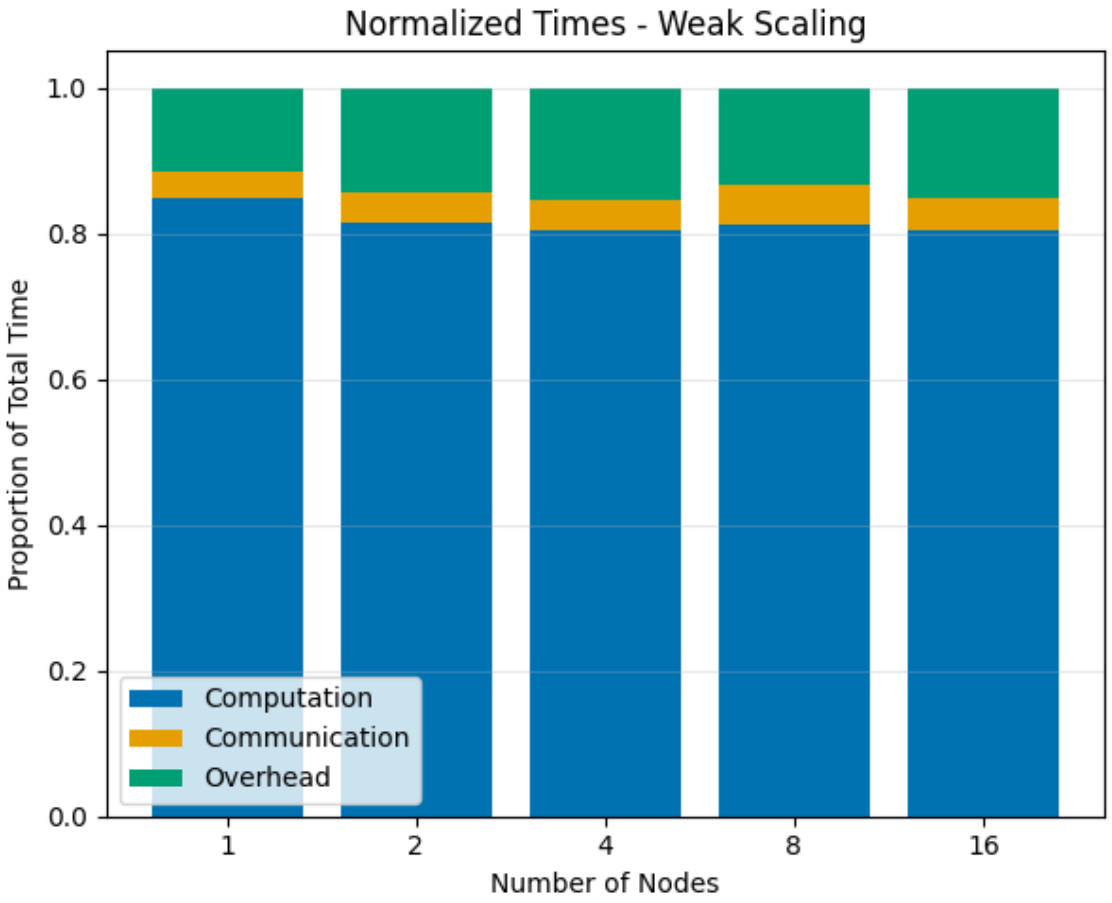
Weak Scaling - Total time





Time spent doing…





Valgrind Analysis

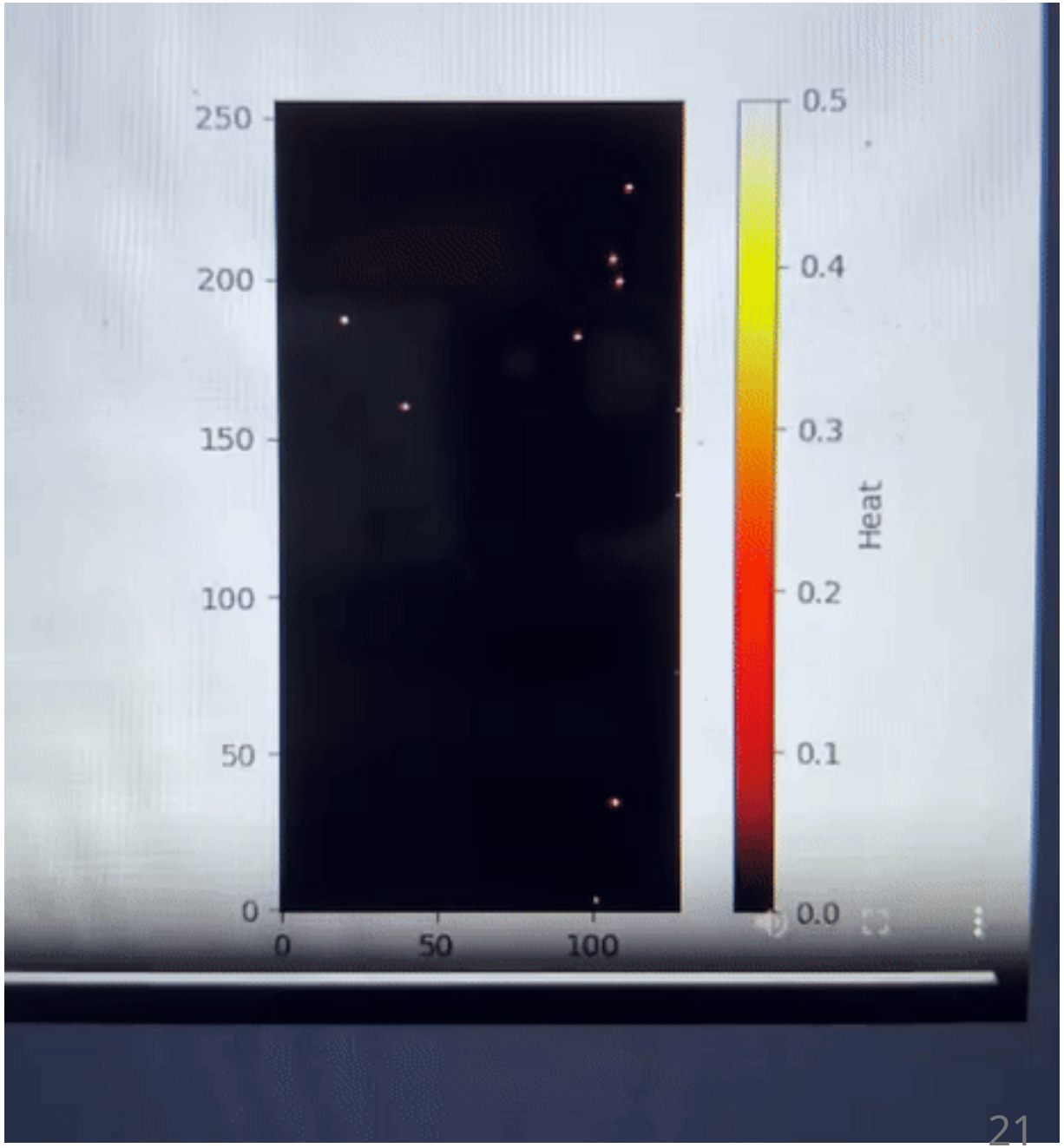
Cache Miss Rates (Cachegrind)

Cache Level	Instructions	Data (Reads)	Data (Writes)	Data (Total)
L1 (L2 run)	0.03%	2.9%	7.8%	4.0%
L1 (L3 run)	0.02%	2.9%	7.8%	4.0%
L2 (LL in run 1)	0.01%	2.3%	7.6%	3.5%
L3 (LL in run 2)	0.00%	0.1%	0.5%	0.2%

Testing and Troubleshooting

- man
- Davide's plot
- GDB
- Assembly

Davide's plot



GDB

```
mpicc -o maindebug -Iinclude src/stencil_template_parallel.c -g
mpiexec -np 2 gnome-terminal --wait -- gdb -x ./gdb_commands ./maindebug
```

Initial frequent problem: *segmentation fault*.

Solved by: backtracing (`bt` command inside of `gdb`)

Assembly

```
mpicc -O3 -march=native -fopenmp -S -fverbose-asm -masm=intel -Iinclude src/stencil_template_parallel.c -o stencil.s
```

```
# src/stencil_template_parallel.c:314:   return old[idx] * 0.5 + (old[idx - 1] + old[idx + 1] +
        vmovsd  xmm0, QWORD PTR [rcx+r13*8]     # *_87, *_87
        mov     rbp, r12        #,
        vaddsd  xmm0, xmm0, QWORD PTR [rcx+r12*8]       # tmp176, *_87, *_82
# src/stencil_template_parallel.c:315:                            old[idx - fxsize] + old[idx + fxsize]) *
        mov     r12d, esi       # i, i
# src/stencil_template_parallel.c:420:     new[IDX(i, 1)] = stencil_computation(old, fxsize, i, 1);         // top border
        lea     edi, [rdx+1]    #,
# src/stencil_template_parallel.c:314:   return old[idx] * 0.5 + (old[idx - 1] + old[idx + 1] +
        vaddsd  xmm0, xmm0, QWORD PTR [rcx+r12*8]       # tmp179, tmp176, *_93
# src/stencil_template_parallel.c:315:                            old[idx - fxsize] + old[idx + fxsize]) *
        lea     r12d, [rbx+rbp] # tmp182,
# src/stencil_template_parallel.c:314:   return old[idx] * 0.5 + (old[idx - 1] + old[idx + 1] +
        lea     r13d, [rbp+0+r8]        # tmp191,
# src/stencil_template_parallel.c:315:                            old[idx - fxsize] + old[idx + fxsize]) *
        vaddsd  xmm0, xmm0, QWORD PTR [rcx+r12*8]       # tmp183, tmp179, *_99
# src/stencil_template_parallel.c:314:   return old[idx] * 0.5 + (old[idx - 1] + old[idx + 1] +
        lea     r12d, [rsi+r9]  # tmp193,
# src/stencil_template_parallel.c:420:     new[IDX(i, 1)] = stencil_computation(old, fxsize, i, 1);         // top border
        mov     rdx, rdi        #,
# src/stencil_template_parallel.c:315:                            old[idx - fxsize] + old[idx + fxsize]) *
        vmulsd  xmm0, xmm0, xmm2        # tmp184, tmp183, tmp206
# src/stencil_template_parallel.c:314:   return old[idx] * 0.5 + (old[idx - 1] + old[idx + 1] +
        vfmadd231sd     xmm0, xmm1, QWORD PTR [rcx+rdi*8]       # _103, tmp207, *_76
# src/stencil_template_parallel.c:420:     new[IDX(i, 1)] = stencil_computation(old, fxsize, i, 1);         // top border
        vmovsd  QWORD PTR [r10+rdi*8], xmm0     # *_29, _103
# src/stencil_template_parallel.c:314:   return old[idx] * 0.5 + (old[idx - 1] + old[idx + 1] +
        vmovsd  xmm0, QWORD PTR [rcx+r13*8]     # *_50, *_50
# src/stencil_template_parallel.c:421:     new[IDX(i, ysize)] = stencil_computation(old, fxsize, i, ysize); // bottom border
        lea     edi, [r11+rbp]  # tmp189,
# src/stencil_template_parallel.c:314:   return old[idx] * 0.5 + (old[idx - 1] + old[idx + 1] +
        vaddsd  xmm0, xmm0, QWORD PTR [rcx+r12*8]       # tmp194, *_50, *_55
```

Thank you!