PTfS-CAM

Project: Modelling 2D steady-state heat equation

Part 2

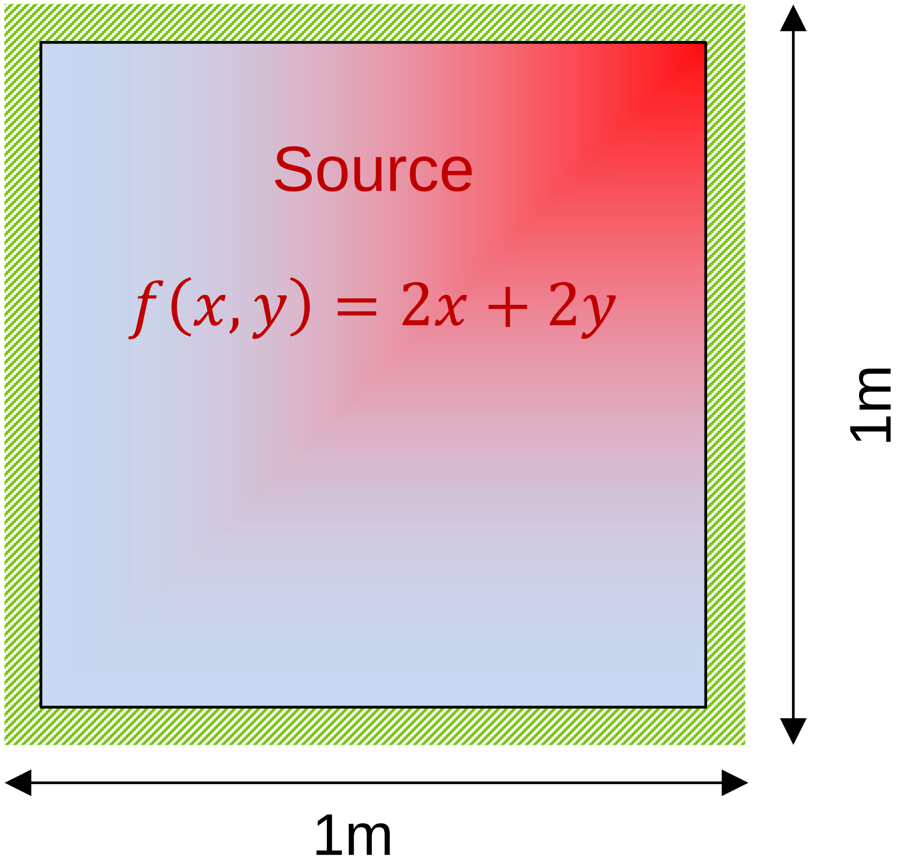
Overview

- Recap from part 1
- Discuss new tasks
- Roofline Model example

Scenario: Heat dissipation on a rectangular plate

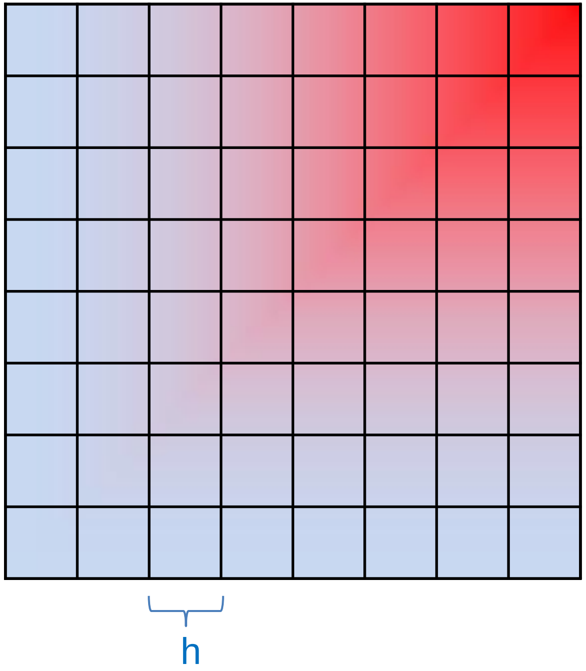
Find steady state temperature distribution inside the plate!

Boundary
$T(\varphi) = 0$



Discretization

$$-\Delta u = -\left(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2}\right) = f$$



Use Finite Difference Method (FDM) for discretization

$$\Rightarrow -\Delta u = -\left(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2}\right)(x, y) \approx$$

$$\frac{1}{h^2}\big(4u(x,y) - u(x-h,y) - u(x+h,y) - u(x,y-h) - u(x,y+h)\big)$$

Solving linear system

Solve for $u : -\Delta u = f$

1. Use Conjugate Gradient (CG)

2. Use Preconditioned Conjugate Gradient (PCG) with symmetric Gauss-Seidel preconditioning

Your tasks

1. Clone the code from Github:
   `git clone https://github.com/RRZE-HPC/PTfS-CAM-Project.git`

2. Build the code using the given Makefile, i.e., just type **`CXX=icpx make`**

3. To switch on LIKWID measurement (for part 2) set the LIKWID flag to 'on', i.e.,
   `LIKWID=on CXX=icpx make`

4. Check for code correctness using the **test** executable: **`./test`**

5. To run the actual code, use the **perf** executable:
   `./perf num_grids_y num_grids_x`

6. If all tests pass, **parallelize building blocks using OpenMP**. Always observe correctness!

7. Are there any possible performance optimizations that you could do in the CG and PCG solver implemented in **`SolverClass::(P)CG`** (Solver.cpp)? If so, implement them!

Your tasks (new)

8. Calculate roofline predictions in [LUP/s] for CG and PCG on 1 ccNUMA domain (18 cores) of Fritz. Calculate for three grid sizes : 2000×20000, 20000×2000 and 1000×400000. The last dimension is in x-direction (innermost). Does the performance change? Why?

9. Check whether you attain the roofline performance by running the code on 1 ccNUMA domain of Fritz for the three dimensions given above. Timings are already included. Run the code using following command.
   ```
   <OMP_stuff> ./perf  num_grids_y  num_grids_x
   ```

10. Measure the code balance in [bytes/LUP] of ‘applyStencil’ and ‘GSPreCon’ kernels on 1 ccNUMA domain (18 threads) for the three grid dimensions and comment whether it agrees with your model. You can use LIKWID for the measurement.
    - To switch on LIKWID measurement set the LIKWID flag to ‘on’, i.e.,
      ```
      LIKWID=on CXX=icpx make
      ```

11. Plot the scalability of CG and PCG from 1 → 4 ccNUMA domains (4 ccNUMA domains = 1 node = 72 cores) of Fritz. Does it scale perfectly? If not, why? Can you fix it?

Performance prediction for entire algorithm

PCG example

```
while( (iter<niter) && (res_norm > tol*tol) )
```

$$
\begin{aligned}
v &= A\,p \\
\lambda &= \frac{\alpha 0}{<v,p>} \\
x &= x + \lambda p \\
r &= r - \lambda v \\
z &= Pr \\
\alpha 1 &= <r,z> \\
p &= z + \frac{\alpha 1}{\alpha 0} p \\
\alpha 0 &= \alpha 1
\end{aligned}
$$

```
++ iter
```

Multiple kernels

Performance prediction for entire algorithm : example

```
void scale(int n,…)
{
    for(int i=0; i<n; ++i)
        a[i] = c*b[i];
}

void add(int n,…)
{
    for(int i=0; i<n; ++i)
        a[i] = b[i] + d*c[i];
}

int main()
{
    …
    scale(1e8,…);
    add(1e8,…);
        …
    return 0;
}
```

Not considering write-allocates

$I_{scale} = \frac{1}{16} \frac{Flop}{Byte}$, $P^{max}_{scale} = 1 * 8 * 2.2 * 18 = 317\ GFlop/s$

flops SIMD f #cores

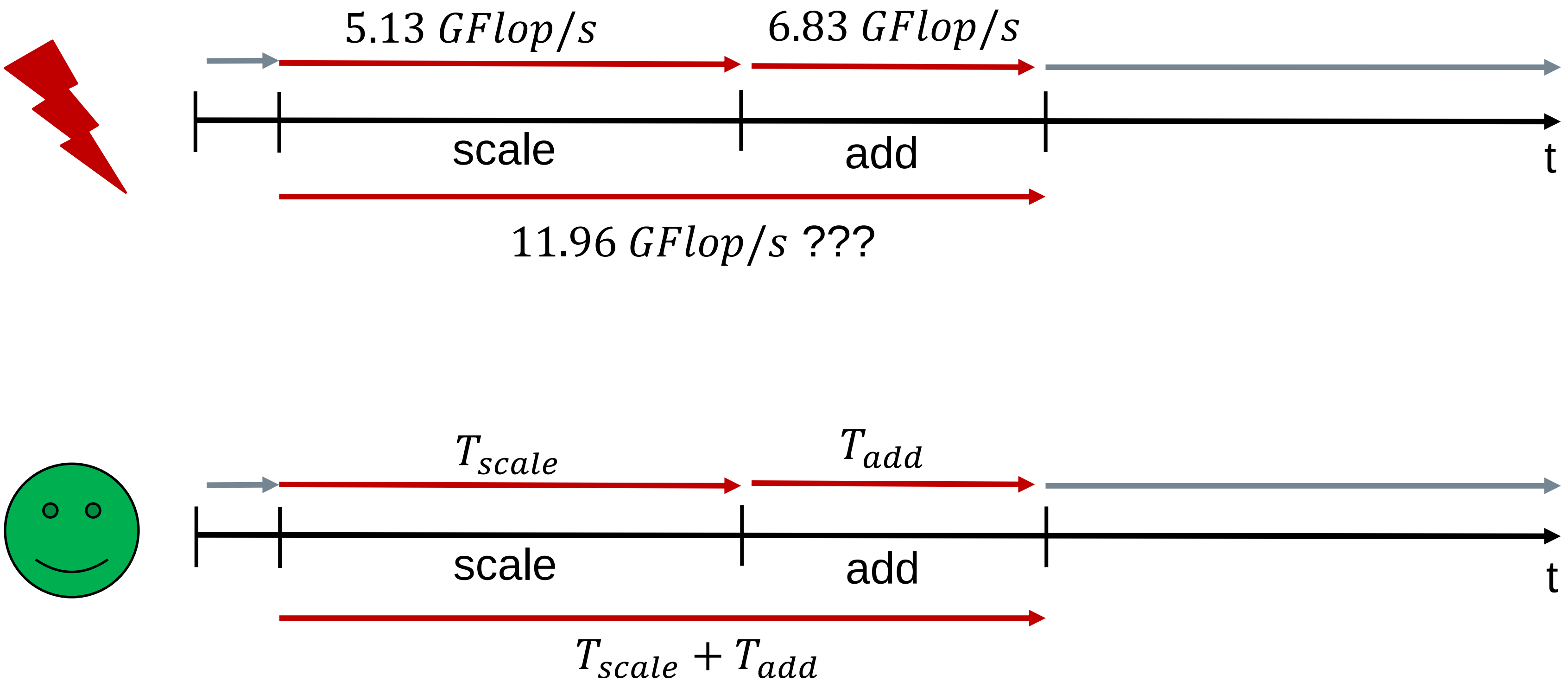
$P_{scale} = \min(317, 1/16*82) = 5.13\ GFlop/s$

$I_{add} = \frac{2}{24} \frac{Flop}{Byte}$, $P^{max}_{add} = 2 * 8 * 2.2 * 18 = 637\ GFlop/s$

$P_{add} = \min(637, 2/24*82) = 6.83\ GFlop/s$

Is it $P_{total} = P_{scale} + P_{add}$?

Performance prediction for entire algorithm : example



Performance prediction for entire algorithm : example

```
void scale(int n,…)
{
    for(int i=0; i<n; ++i)
        a[i] = c*b[i];
}

void add(int n,…)
{
    for(int i=0; i<n; ++i)
        a[i] = b[i] + d*c[i];
}

int main()
{
    …
    scale(1e8,…);  }
    add(1e8,…);    }
        …
    return 0;
}
```

$I_{scale} = \frac{1}{16} \frac{Flop}{Byte}$, $P_{scale}^{max} = 1 * 8 * 2.2 * 18 = 317\ GFlop/s$

$P_{scale} = \min(317, 1/16*82) = 5.13\ GFlop/s$

$I_{add} = \frac{2}{24} \frac{Flop}{Byte}$, $P_{add}^{max} = 2 * 8 * 2.2 * 18 = 637\ GFlop/s$

$P_{add} = \min(637, 2/24*82) = 6.83\ GFlop/s$

Is it $P_{total} = P_{scale} + P_{add}$? **NO**

But $T_{total} = T_{scale} + T_{add}$

$T_{total} = \frac{1e8*1\ Flop}{P_{scale}} + \frac{1e8*2\ Flop}{P_{add}} = 0.0487$

$\Rightarrow P_{total} = \frac{1e8\ It}{T_{total}} = 2.05\ GIT/s$

A more useful measure of performance here

Things to take care

- Think to use loop fusion wherever necessary.
- For debugging please compile code as: **`CXX=icpx make EXTRA_FLAGS=-DDEBUG`**
- Sometimes it's useful for debugging to visualize your arrays. Use the function **`writeGnuplotFile`** and plot using **`splot`** in gnuplot if needed.
- Take particular care with parallelizing the Gauss-Seidel preconditioner. Recall the "Wavefront" parallel scheme from lecture.
- Use Fritz (Ice Lake) for getting your performance results.
- Fix clock frequency to 2.2 GHz (and use **`performance`** governor)
- Check if the measurements are reproducible (i.e. pinning, scheduling, and clock frequency).
- Request a dedicated compute node for measuring performance.

Things to take care

- Remember pinning (`-C` with **`likwid-perfctr`**) when doing performance counter measurements. Use `-m` for using markers, instead of end-to-end measurement.
- When measuring parallel loops with **`likwid-perfctr`**, LIKWID_MARKER_START and LIKWID_MARKER_STOP should be called by all threads.
- Remember **`likwid-perfctr`** can incur some overheads, so for performance measurement better run without **`likwid-perfctr`**.
- Remember to arrive at final roofline model of (P)CG, you would need to stitch performance models of different kernels. It might be convenient to use a time-based model (see previous slides). If you wish use Excel sheet for this.
- For roofline modeling you can assume the memory bandwidth of 1 ccNUMA domain of Fritz is 82 GB/s.
- Remember Fritz has a write-allocate avoiding mechanism that can kick in some cases.

Submission

- Submit the code after doing tasks using the link in PTfS Moodle. Please compress and submit only a single (ZIP) file. Include code and the report in submission.
- Deadline is 1 week before your oral exam.
- The report should contain all the details necessary to reproduce your measurements.
- All submitted code should be compilable by just typing **`make`** (Makefile is already provided),
- Your code should pass all the tests.
- Both executables (**`test`** and **`perf`**) should run without segmentation faults.
- While submitting report expected roofline performance and the measured performance, use $\left[\frac{LUP}{s}\right]$ (= $\left[\frac{IT}{s}\right]$) as performance metric, see definition in **`perf.cpp`**.
- If there is any substantial deviation between these values, please explain plausible cause if any.

Scoreboard (optional)

- Submit your best run on Moodle (see “PTfS-CAM Project Leaderboard”) to see who’s the fastest!
- See instructions on the submission page
- Final submission: End of semester (Sept 30)
- The best coding project(s) win(s) a prize!

Final remark

- In the exam you will be definitely asked questions based on this exercise.

- Happy Coding !!!