LUDWIG-
MAXIMILIANS-
UNIVERSITÄT
MÜNCHEN

Prof. Dr. D. Kranzlmüller, Dr. K. Fürlinger

Parallel Computing

WS 2017/18

Session 12: Scalability, PGAS

Tobias Fuchs, M.Sc.
tobias.fuchs@nm.ifi.lmu.de

Scalability

Recap

Performance tuning

Improve time to completion / throughput / latency / … achieved on a single computational unit

Evaluated system is arbitrary but its configuration does not change in benchmark scenarios

Hardware Scalability

Improve speedup achieved when adding computational components to the system

Scalability measures in general:

Extending computational capacities by a factor k ideally reduces time to completion by factor k (linear speedup).

Strong Scaling

Increase degree of hardware parallelism

Observe speedup for **fixed total problem size** (usually number of elements)

Weak Scaling

Increase degree of hardware parallelism

Observe speedup for **fixed problem size *per processor***

Scalability measures in general:

Extending computational capacities by a factor k ideally reduces time to completion by factor k (linear speedup).

Strong Scaling → expose Amdahl's Law

Increase degree of hardware parallelism

Observe speedup for **fixed total problem size** (usually number of elements)

Weak Scaling → expose Gustafson's Law

Increase degree of hardware parallelism

Observe speedup for **fixed problem size *per processor***

Scalability measures in general:

Extending computational capacities by a factor k ideally reduces time to completion by factor k (linear speedup).

Amdahl's Law

$$\boldsymbol{Speedup = \frac{1}{seq + (1 - seq)/p}}$$

Gustafson's Law

$$\boldsymbol{Speedup = \frac{(s + p \cdot N)}{s + p}}$$

Matrix Product with OpenMP

```
void mmult_naive_par(double A[M][N], double B[N][K], double C[M][K]) {
    int    i, j, k;
    double sum;
    #pragma omp parallel for private(j,k,sum)
    for (i = 0; i < M; i++) {
        for (j = 0; j < K; j++) {
            sum = 0.0;
            for (k = 0; k < N; k++) {
                sum += A[i][k] * B[k][j];
            }
            C[i][j] = sum;
        }
    }
}
```

Matrix Product with OpenMP

N/T	1	2	4	8	16	32
100	0.89	0.78	0.82	0.50	0.12	0.11
1000	14.12	19.90	32.91	32.32	26.22	21.66
2000	14.30	28.74	42.63	64.76	65.04	35.31
4000	14.68	29.09	57.93	82.64	156.84	138.94

These are actual measurements submitted for a highly optimized implementation of mmult.

Spot the effects of

- Amdahl’s Law
- Gunther’s Law
- Gustafson’s Law

Matrix Product with OpenMP

N/T	1	2	4	8	16	32
100	0.89	0.78	0.82	0.50	0.12	0.11
1000	14.12	19.90	32.91	32.32	26.22	21.66
2000	14.30	28.74	42.63	64.76	65.04	35.31
4000	14.68	29.09	57.93	82.64	156.84	138.94

Spot the effects of

- Amdahl's Law

Performance increase (speedup) is **limited by sequential sections and degree of parallelism**.

Note that super-linear speedup is rare but possible!

Matrix Product with OpenMP

N/T	1	2	4	8	16	32
100	0.89	0.78	0.82	0.50	0.12	0.11
1000	14.12	19.90	32.91	32.32	26.22	21.66
2000	14.30	28.74	42.63	64.76	65.04	35.31
4000	14.68	29.09	57.93	82.64	156.84	138.94

Spot the effects of

- Gunther's Law

Performance may **worsen** if degree of parallelism is increased because contention rate increases.

Matrix Product with OpenMP

N/T	1	2	4	8	16	32
100	0.89	0.78	0.82	0.50	0.12	0.11
1000	14.12	19.90	32.91	32.32	26.22	21.66
2000	14.30	28.74	42.63	64.76	65.04	35.31
4000	14.68	29.09	57.93	82.64	156.84	138.94

Spot the effects of

- Gustafson's Law

Higher degree of parallelism can yield performance benefit **when problem size is increased**. For example:

N = 1000 → GLFOPS drop from 8 to 16 threads.

N = 2000 → GLFOPS saturated from 8 to 16 threads.

N = 4000 → GFLOPS increased from 8 to 16 threads.

Matrix Product with OpenMP

N/T	1	2	4	8	16	32
100	0.89	0.78	0.82	0.50	0.12	0.11
1000	14.12	19.90	32.91	32.32	26.22	21.66
2000	14.30	28.74	42.63	64.76	65.04	35.31
4000	14.68	29.09	57.93	82.64	156.84	138.94

Spot the effects of

- Gunther’s Law + Gustafson’s Law

Speedup saturation as predicted by Amdahl’s and Gustafson’s models **shifted** to higher degrees of parallelism **with increasing problem size**.

N = 1000 → Saturates with 4 threads

N = 2000 → Saturates with 8 threads

N = 4000 → Saturates with 16 threads

PGAS

Experimenting with the DASH C++ Library

Using DASH

Clone DASH from github:

```
$ git clone https://github.com/dash-project/dash.git ./dash
```

Copy to your SuperMUC home directory:

```
$ scp -r ./dash <user>@hw.supermuc.lrz.de:~/
```

Build DASH on SuperMUC in your home directory:

```
$ ssh <user>@hw.supermuc.lrz.de
$ cd dash
(~/dash) $ ./build.sh
```

Using DASH

Run DASH examples on SuperMUC:

```
(~/dash) $ cd build
(~/dash/build) $ mpirun -n 8 ./bin/ex.01.hello.mpi
```

Tobias Fuchs

tobias.fuchs@nm.ifi.lmu.de

www.mnm-team.org/~fuchst