Written exam

Day: August 9th, 2019

Time: 10:30 a.m.

Location: H8

General information

- **Time:**
  - 5 ECTS (lecture only): 60 minutes
  - 7.5 ECTS (lecture + tutorials): 90 minutes
  - Same questions for both + additional questions (covering tutorials) for 7.5

- **Permitted aids**
  - Pen & ruler
  - Simple (non-programmable) calculator
  - YOUR BRAIN

- **All code will be given in C**

- **General motivation of exam questions:**
  Understand basic architectural concepts, relevant code transformations and the interaction of both resulting in actual performance (making sense of performance)

What to learn? Basic concepts

- Do not try to learn numbers and terms without understanding them – example:
- Question 1.a.: Parallelize the following subroutine using OpenMP

```
double dmvm(int n, int m, double *lhs, double *rhs, double *mat, double
sum){
double s=0.;
for(r=0; r<n; ++r) {
    offset=m*r;
    for(c=0; c<m; ++c)
      lhs[r] += mat[c + offset]*rhs[c];
    s = s + lhs[r]*rhs[r];
}
return sum+s;
}
```

Perfect answer

```
double dmvm(int n, int m, double *lhs, double *rhs, double *mat,
            double sum){
double s=0.;
#pragma omp parallel for private(offset,c) reduction(+:s)
for(r=0; r<n; ++r) {
    offset=m*r;
    for(c=0; c<m; ++c)
      lhs[r] += mat[c + offset]*rhs[c];
    s = s + lhs[r]*rhs[r];
}
return sum+s;
}
```

Also valid/good answer

```
double dmvm(int n, int m, double *lhs, double *rhs, double *mat,
            double sum){

double s=0.;

#pragma omp parallel for private(offset,c) reduction*

for(r=0; r<n; ++r) {

    offset=m*r;

    for(c=0; c<m; ++c)

      lhs[r] += mat[c + offset]*rhs[c];

    s = s + lhs[r]*rhs[r];
}

return sum+s;

}

*: I do not know exact syntax but I need a reduction operation on sum
```

What to learn? Basic concepts

- Question 1.b.: What will be the maximum performance of this code for n=m=20,000 on a multicore processor chip with b_S=48 GB/s bandwidth and an L3 cache of 20 MiB, assuming that the code is memory bound?

What to learn? Basic concepts

- Question 1.c.: Your OpenMP parallel subroutine is called from a code which runs on a 2-socket node of ccNUMA architecture. Assume large n and m: Does performance always perfectly scale from 1 to 2 sockets?

```
double dmvm(int n, int m, double *lhs, double *rhs, double *mat,
            double sum){

double s=0.;

#pragma omp parallel for private(offset,c) reduction(+:s)

for(r=0; r<n; ++r) {

    offset=m*r;

    for(c=0; c<m; ++c)

      lhs[r] += mat[c + offset]*rhs[c];

    s = s + lhs[r]*rhs[r];
}

return sum+s;

}
```

What to learn? Basic concepts

- Question 1.d.: Optimize single core execution for a minimum L2 code balance, assuming an L1 cache of 32 KiB.

```
double dmvm(int n, int m, double *lhs, double *rhs, double *mat,
            double sum){

double s=0.;

for(r=0; r<n; ++r) {

    offset=m*r;

    for(c=0; c<m; ++c)

      lhs[r] += mat[c + offset]*rhs[c];

    s = s + lhs[r]*rhs[r];
}

return sum+s;

}
```

What to learn? Basic concepts

- Question 2.a.: Briefly describe the first touch concept in ccNUMA architectures
- Question 2.b.: What implication does this concept have on shared memory parallel *programming* for ccNUMA nodes if we aim for optimal performance?
- Question 3.a.: Briefly describe the concept of superscalarity!
- Question 3.b.: Name one performance metric that quantifies the level of superscalarity in a running code!
- Question 3.c.: Given an architecture (specs) – can the following code fully exploit superscalarity?

What to learn – basic concepts

- For a given code and architectural description build the *refined/extended* roofline model

- Draw the *basic* roofline graph for this machine: clock speed, FMA, SIMD, cores, memory bandwidth

- How does it change if SIMD is disabled....?

- --------

- Performance of a simple benchmark as a function of loop length is given: Determine how many cache levels & of which size?

Questions

- If you run that OpenMP parallel code, what do you need to consider to get reliable / useful performance numbers
  - Exclusive / Clock speed / Affinity-pinning / variations / reasonable-sensibility

A selection of very important slides

- **Slideset 2:**
  - Performance and work metrics (02/3-7)
  - Impact factor & best practices for performance measurements (02/20-24)

- **Slideset 3:**
  - Basic resource bottlenecks of code execution (03/13-14)
  - Pipelining (03/23-45, NOT 34)
  - Superscalarity & OOO (03/52-55)
  - SIMD (03/59-74)
  - Performance composition (03/76-78)
  - P_max calculation (03/79-88)

A selection of very important slides

- **Slideset 4:**
  - Effective BW model (!!) (04/5-7)
  - Spatial locality & memory layout (04/14-19)
  - Temporal locality (04/19,20)
  - Basics of cache mapping (direct, m-may set-assoc.)
  - Cache mgmt. details (04/35-40)
- **Slideset 5:**
  - cache sizes and LD/ST throughput (05/10,12)
  - dMVM traffic analysis (05/13-28)
  - Algorithm classification (05/30-39)

A selection of very important slides

- **Slideset 6+7:**
  - Multicore (0607/3,7)
  - Shared memory (0607/12-19)
  - Parallel vs. shared resources (0607/21-24)
  - **Not relevant for exam: Cache coherence & False Sharing**

- **Slideset 8:**
  - Everybody should be able to parallelize a simple kernel with OpenMP „correctly" and achieve good performance on modern architectures
  - Basic ideas, keywords, concepts
  - **Not relevant for exam: ordered, locks, threadprivate**

- **Slideset 9:**
  - Roofline (RL) & extended RL: Immensely important! (09/9-end)

A selection of very important slides

- **Slideset 10: Jacobi – important!**
  - RL analysis, Layer Conditions (!), LCs & shared caches
  - Overhead with middle loop parallelization
- **Slideset 11: spMVM (not so important)**
  - Basic idea of CRS format and implementation
  - Minimum code balance for spMVM
- **Slideset 12: advanced OpenMP**
  - Race conditions, deadlocks
  - OpenMP overheads (12/3-6,8,10)
  - ccNUMA data placement (12/18-29)
- **Slideset 13: Parallelism**
  - Amdahl’s & Gustafson’s Laws, incl. communication overhead

A selection of very important slides

- **Slideset 14: Networks**
  - Transfer time model (14/7)
  - Ping-Pong benchmark
  - Bisection bandwidth
  - Bus & switched networks (fat tree)

- **Slideset 15:**
  - MPI Point-to-Point communication:
    - Blocking vs. Non-blocking
    - Asynch. Vs synch.
    - DO not learn library calls but understand concepts

Additional material from the tutorials

- **Power dissipation issues**
  - f^3 law for dynamic power
  - Multicore energy model (baseline power, dynamic power)
  - Relevance of performance saturation for energy consumption

- **Slow computing**
  - Machine with slower CPUs scales better
  - Slow code scales better
  - Why?

- **Speedup comparisons**
  - How can we predict the speedup of a code between architectures?
  - Issues with massively parallel runs?
  - The role of “sensible parallel efficiency”