Introduction to OpenMP

- Goal: make parallel programming easy
- Shared memory programming
- Compiler pragma for C/C++ and Fortran
  - Requires a compatible compiler
- Require a runtime helper library
- Portable solution
- Official site: http://openmp.org/

Parallel section example

#pragma omp: compiler directive

API OpenMP

```
/*
 * Example 1: omp parallel
 */
qDebug() << "omp parallel begin";
#pragma omp parallel
{
    qDebug() << "rank" << omp_get_thread_num() <<
                "size" << omp_get_num_threads();
}
qDebug() << "omp parallel end";
```

There is an implicit barrier at the end of a parallel scope

Chaque fil d'exécution va exécuter le bloc de code.

pragma

- #pragma omp directive [clause, ...]
- Applies to a scope { ... }
  - or the following statement
- Directives:
  - parallel, critical, single, barrier, atomic
- Clauses:
  - for, private, reduction

API OpenMP

- Utilities (optional)
- #include <omp.h>
- Examples:
  - omp_get_num_threads(): returns the number of started threads
  - omp_get_thread_num() returns the thread rank

Environnement variables

- Can modify the behavior without recompiling
- Example:
  - OMP_WAIT_POLICY: passive or active wait on a barrier
  - OMP_NUM_THREADS: number of threads to spawn (default is the number of cores)

Compiling OpenMP program

- GCC: -fopenmp (man gcc)
- Compile: QMAKE_CXXFLAGS += -fopenmp
- Link: QMAKE_LFLAGS += -fopenmp
- warning: ignoring #pragma omp parallel [-Wunknown-pragmas]
- error: undefined reference to `omp_get_num_threads'

Parallel computation

```
auto memset_serial = [&]() {
    for (int i = 0; i < n; i++) {
        v[i] = i * i;
    }
};

auto memset_parallel = [&]() {
    #pragma omp parallel for
    for (int i = 0; i < n; i++) {
        v[i] = i * i;
    }
};
```

#pragma omp parallel:
spawns a parallel section

#pragma omp for:
Split loop iterations

Private or shared variables?

```
// variables
int n = 1000;
float max = 0.0;
int i = 0;
float p = 0.1;
float t = 0.0;
QVector<float> orig(n);
QVector<float> norm(n);

// init sound-like data
#pragma omp parallel for
for (i = 0; i < n; i++) {
    orig[i] = std::cos((t + 0.1) * 0.1) +
              std::cos((t + 0.2) * 0.2) +
              std::cos((t + 0.3) * 0.3);
    t += p;
}
```

Parent scope: shared by default

One exception: loop index is private by default

Beware: t is shared and subject to race condition. Must protect it with mutex (or #pragma omp critical) or use private(t) or some reduction operator.

Variable scope

- Clause shared: shared by default, except loop index
- Clause private: thread local
- Clause firstprivate: thread local, but initialized with the content of the parent scope
- Also available: lastprivate, copyprivate

Critical section

```
// find the max value (manual reduction)
float max = 0.0;
float my_max = 0.0;
#pragma omp parallel private(my_max)
{
    #pragma omp for
    for (i = 0; i < n; i++) {
        my_max = std::max(my_max, std::abs(norm[i]));
    }
    #pragma omp critical
    {
        max = std::max(max, my_max);
    }
}

qDebug() << "check max=" << max;
```

Variable my_max private to each thread

Safe update of shared variable.

Reduction

Variable the_max shared,
but OpenMP creates a local
variable for intermediate reduction

```
// find the max value (automatic reduction)
float the_max = 0.0;
#pragma omp parallel for reduction(max:the_max)
for (i = 0; i < n; i++) {
    the_max = std::max(the_max, std::abs(norm[i]));
}

qDebug() << "check max=" << the_max;
```

Implicit critical section for the final reduction.

Operators for reduction (initialization values)			
+	(0)	|	(0)
*	(1)	^	(0)
-	(0)	&&	(1)
&	(~0)	||	(0)
max (Least representable number in **reduction** list item type)			
min (Largest representable number in **reduction** list item type)			

Source: OpenMP 4.0 API C/C++ Syntax Quick Reference Card

Independent computation: nowait

```
#pragma omp parallel
{
    #pragma omp for nowait
    for (int i=0; i<n; i++) a[i] = b[i] + c[i];

    #pragma omp for nowait
    for (int i=0; i<n; i++) d[i] = e[i] + f[i];

    #pragma omp barrier
    scale = sum(a,0,n) + sum(d,0,n);
}
```

Explicit barrier required to wait for all previous computation to finish

The first and the second loops execute at the same time

Examples

- 19-omp-base: pragma examples
- 20-omp-speedup: Measure speedup obtained with OpenMP
- 21-omp-scope: Examples of variable scope
- 22-omp-seq: Using OpenMP to normalize sound signal
- 23-omp-task: Traverse tree-like structure in parallel
- 24-omp-atomic: Example of pragma atomic

Performance counters

- Counts microarchitecture events
- Cache accesses and misses
  - load, store
  - L1, Last Level Cache (LLC)
  - data, instruction
- Can compute cache misses ratio
- Can compute Cycle / Instructions = CPI

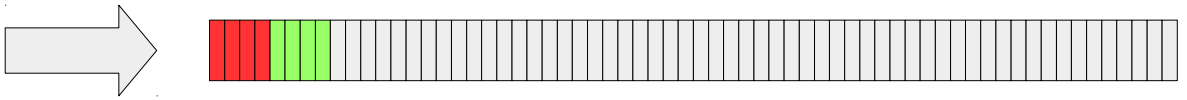
Cache misses

- Linear access cold start
  - Few first accesses causes cache misses, then hardware pre-fetcher will populate the cache in advance
- Large stride in memory (accessing 2D data by column)
- Random access
- Data size v.s. size of the cache: *working set size*
- Sharing a cache line with other cores
  - False sharing and cache line bouncing (lock)

False sharing

```
typedef struct point {
    int x;
    int y;
} point_t;

point_t *foo;
```



Modification (CPU 0)

```
// moving in an airplane
// wheeee!

for (;;)
    foo->x++;
```

Read (CPU 1)

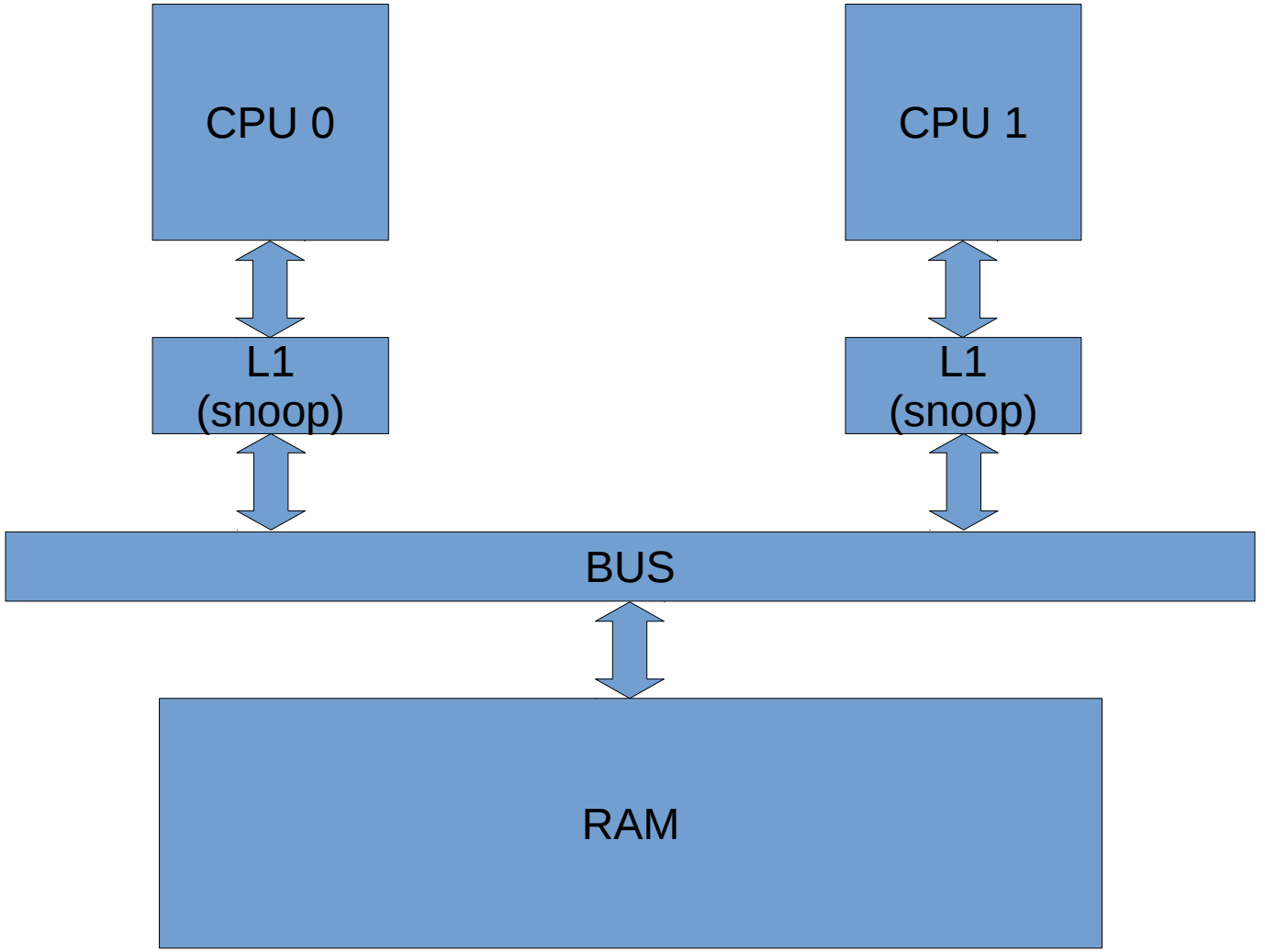
```
// if latitude is close to the equator
// then you are lucky

bool is_lucky(point_t &p) {
    return (foo->y > -20 &&
        foo->y < -20);
}

for (;;)
    is_lucky(foo);
```

Simultaneous Multithreading (SMT)

Intel: HyperThreading

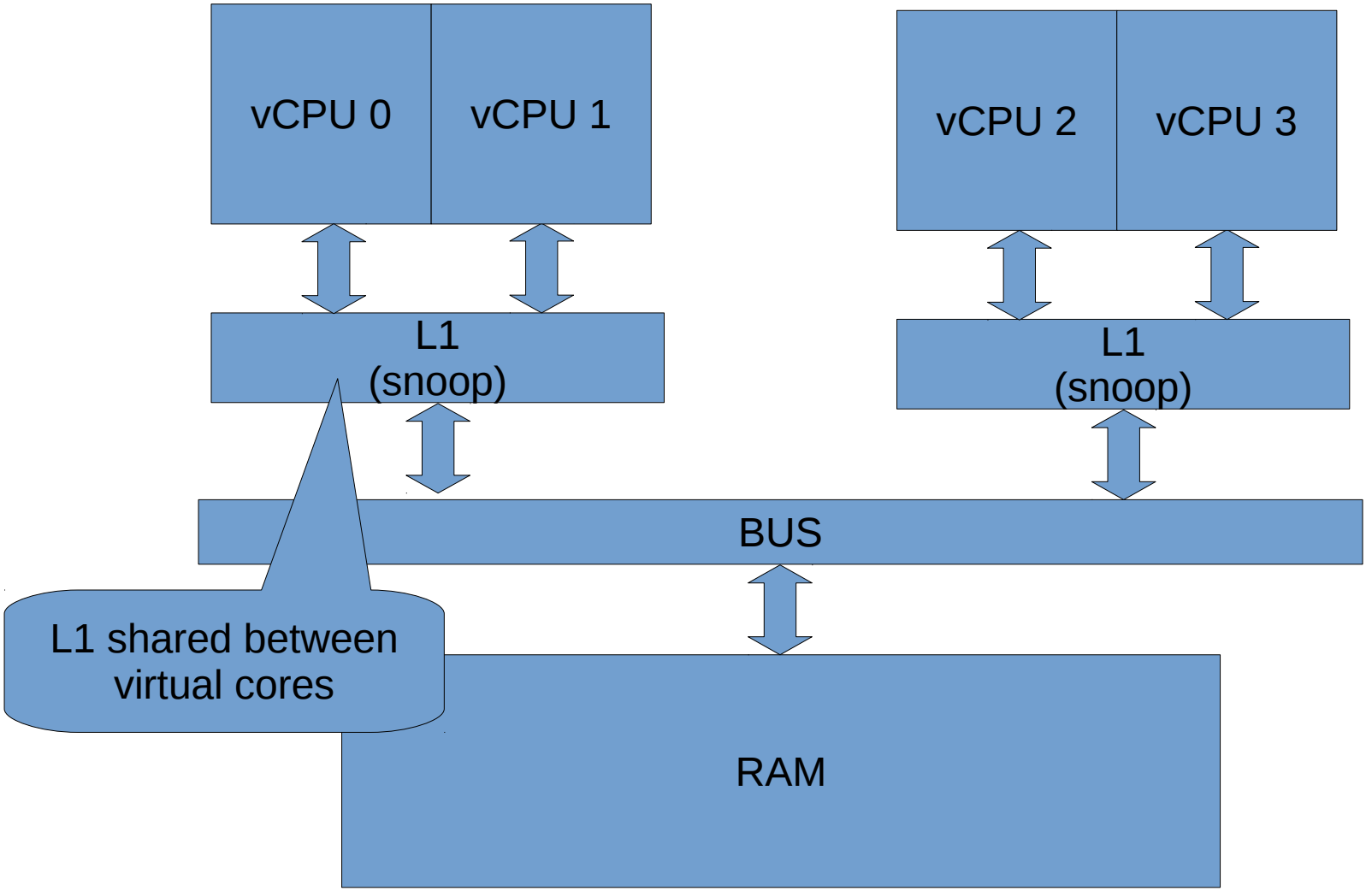


POLYTECHNIQUE MONTRÉAL

ERICSSON

Simultaneous Multithreading (SMT)

Intel: HyperThreading



POLYTECHNIQUE MONTRÉAL

ERICSSON

Activity encode

- Encode applies rot-13 on a buffer and compute the checksum
- Implement the encode_fast() with OpenMP
- Run benchmarks:
  - ./perf-encode.sh pmu
  - ./perf-encode time
- Load the data in the spreadsheet encode.ods
  - Each result file should be copied into the corresponding sheet based on its name
- Analyze the data