Parallel computing platforms

Jeremy Iverson

College of Saint Benedict & Saint John’s University

plan for the day

- Naive Bayes assignment updates (40min + 5min break)
- parallel computer organization (40min + 5min break)
- classes of parallel computation (40min + 5min break)
- our first parallel algorithms (40min + 5min break)

Naive Bayes in C

parallel computer organization

- in order to thinking about how we might solve problems using parallel computation, we need to understand what parallel capabilities our computing systems have.
- there are two fundamental questions we need to ask…
  - control mechanism — how are instructions executed in parallel
  - communication model — how do processing units communicate

parallel computer organization

- control mechanism

- in order to thinking about how we might solve problems using parallel computation, we need to understand what parallel capabilities our computing systems have.
- there are two fundamental questions we need to ask...
  - control mechanism — how are instructions executed in parallel
  - communication model — how do processing units communicate

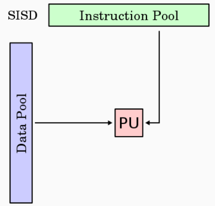
parallel computer organization

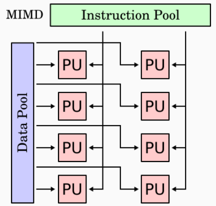
- control mechanism
- communication model

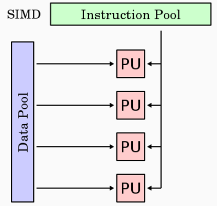
- in order to thinking about how we might solve problems using parallel computation, we need to understand what parallel capabilities our computing systems have.
- there are two fundamental questions we need to ask...
  - control mechanism — how are instructions executed in parallel
  - communication model — how do processing units communicate

flynn’s taxonomy

- based on the number of instruction streams and data streams available in the architecture



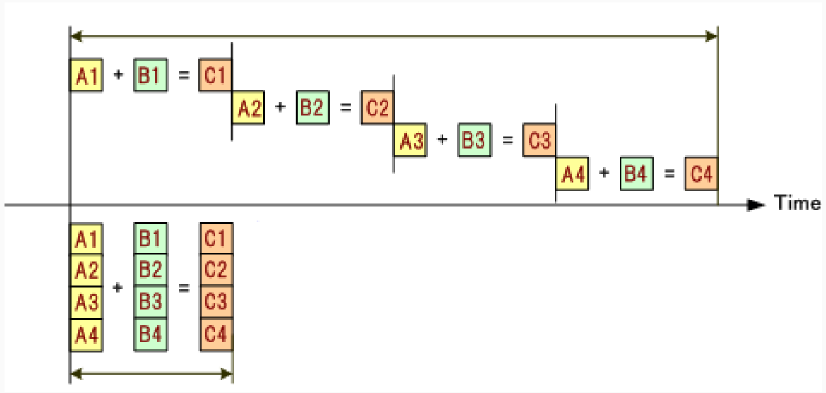




Flynn’s taxonomy by Cburnett / CC BY 3.0 / presenting the four together

- ask students to give examples of each
  - SISD
    - serial computer
  - SIMD
    - gpu
    - many modern cpus have simd extensions
  - MISD
    - uncommon, but could be used for things like fault tolerance
  - MIMD (in many cases, this is further divided into a category commonly known as SPMD, single program multiple data)
    - most common type of parallel computer (multi-thread or multi-node)
- which of these might be interesting to us, i.e., which might make good parallel systems?

simd

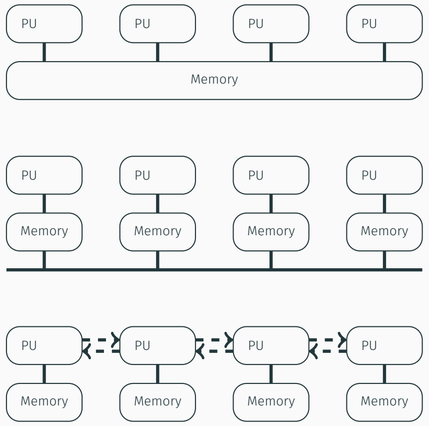


SIMD / cropped from original

- cuda / opencl are languages to express this type of parallelism, but we will not be studying them
- what sorts of problems might this be good for?
  - matrix-oriented scientific computing
  - media-oriented image and sound processors
- simd is simpler and more energy efficient than mimd, why?
  - only needs to fetch one instruction per “cycle”
- what would the mimd time line look like?
- so then what are the (dis)advantages of each of the classifications?
  - simd works in lock stop, so does not require synchronization, which can make it easier to reason about
  - mimd processing elements are autonomous, so can solve more complex problems

communication models

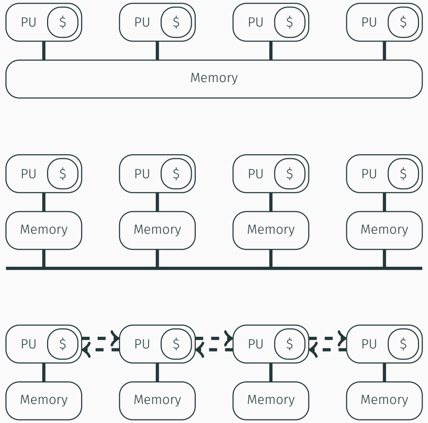
- shared-address space
- message-passing



- how do processing elements communicate with each other
  - shared-address space
    - everyone has access to everyone else's data
    - communicate through shared memory
  - message-passing
    - communicate through messages sent between PUs
    - build on two basic operations, *send* and *receive*
- why not just make everything uma?
- is there anything missing from this model?
  - what is a cache for?
- the existence of caches means that data can exist in multiple places
  - a cache coherence protocol is required to ensure proper semantics and correct program execution
  - have students think of a sequence of operations that would be incorrect without cache coherence

communication models

- shared-address space
- message-passing



- how do processing elements communicate with each other
  - shared-address space
    - everyone has access to everyone else's data
    - communicate through shared memory
  - message-passing
    - communicate through messages sent between PUs
    - build on two basic operations, *send* and *receive*
- why not just make everything uma?
- is there anything missing from this model?
  - what is a cache for?
- the existence of caches means that data can exist in multiple places
  - a cache coherence protocol is required to ensure proper semantics and correct program execution
  - have students think of a sequence of operations that would be incorrect without cache coherence

classes of parallel computation

- With this understanding of parallel computation, we can think about how to formulate algorithms that expose some aspect of parallelism. Typically we will find ourselves in one of two situations, either:
  - we have a bunch of different tasks to carry out, all of which can be done simultaneously
  - we want to do the same task to a bunch of different inputs

  it isn't always obvious which we have and sometimes the same problem can be expressed many ways.
- Is this an example of SIMD or MIMD, neither or both? Why?

classes of parallel computation

Imagine you are the head chef responsible for preparing a wedding banquet. The meal will have four courses: appetizer, salad, main course, and dessert. You have P chefs at under your supervision.

- With this understanding of parallel computation, we can think about how to formulate algorithms that expose some aspect of parallelism. Typically we will find ourselves in one of two situations, either:
  - we have a bunch of different tasks to carry out, all of which can be done simultaneously
  - we want to do the same task to a bunch of different inputs

  it isn't always obvious which we have and sometimes the same problem can be expressed many ways.
- Is this an example of SIMD or MIMD, neither or both? Why?

classes of parallel computation

Imagine you are the head chef responsible for preparing a wedding banquet. The meal will have four courses: appetizer, salad, main course, and dessert. You have *P* chefs at under your supervision.

How best to get the meal served to the guests as quickly as possible?

- With this understanding of parallel computation, we can think about how to formulate algorithms that expose some aspect of parallelism. Typically we will find ourselves in one of two situations, either:
  - we have a bunch of different tasks to carry out, all of which can be done simultaneously
  - we want to do the same task to a bunch of different inputs

  it isn't always obvious which we have and sometimes the same problem can be expressed many ways.
- Is this an example of SIMD or MIMD, neither or both? Why?

classes of parallel computation

Imagine you are the head chef responsible for preparing a wedding banquet. The meal will have four courses: appetizer, salad, main course, and dessert. You have P chefs at under your supervision.

How best to get the meal served to the guests as quickly as possible?

- Each chef works on N/P meals independently of the others.
- Each chef works on a different task related to the preparation of a meal, i.e., cutting carrots, cooking soup, icing cake, etc.

- With this understanding of parallel computation, we can think about how to formulate algorithms that expose some aspect of parallelism. Typically we will find ourselves in one of two situations, either:
  - we have a bunch of different tasks to carry out, all of which can be done simultaneously
  - we want to do the same task to a bunch of different inputs

  it isn't always obvious which we have and sometimes the same problem can be expressed many ways.
- Is this an example of SIMD or MIMD, neither or both? Why?

classes of parallel computation

Imagine you are the head chef responsible for preparing a wedding banquet. The meal will have four courses: appetizer, salad, main course, and dessert. You have P chefs at under your supervision.

How best to get the meal served to the guests as quickly as possible?

- *data parallelism*
- *task parallelism*

- With this understanding of parallel computation, we can think about how to formulate algorithms that expose some aspect of parallelism. Typically we will find ourselves in one of two situations, either:
  - we have a bunch of different tasks to carry out, all of which can be done simultaneously
  - we want to do the same task to a bunch of different inputs

  it isn't always obvious which we have and sometimes the same problem can be expressed many ways.
- Is this an example of SIMD or MIMD, neither or both? Why?

y := a*x + y

```c
void
saxpy(size_t const n,
      float  const a,
      float  const * const x,
      float        * const y)
{
  for (size_t i = 0; i < n; i++) {
    y[i] = a * x[i] + y[i];
  }
}
```

reduction

```
float
min(size_t const n,
    float * const x,
{
  /* x[0] will always contain the current
   * minimum */
  for (size_t i = 0; i < n; i++) {
    if (x[i] < x[0]) {
      x[0] = x[i];
    }
  }
  return x[0];
}
```

reduction cont'd

```c
float
min(size_t const n,
    float * const x,
{
  for (size_t i = 1; i < n; i *= 2) {
    /* x[j] will always contain the current
     * minimum in interval [j,j+2i)*/
    for (size_t j = 0; j < n; j += 2 * i) {
      if (x[j + i] < x[j]) {
        x[j] = x[j + i];
      }
    }
  }
  return x[0];
}
```

sorting

```
count = array of k+1 zeros
for x in input do
    count[x] += 1

total = 0
for i in 0, 1, ... k do
    counti = count[i]
    count[i] = total
    total += counti

for x in input do
    output[count[x]] = x
    count[x] += 1

return output
```