Review for the Final Exam

1. The Final Exam
   - on Tuesday 10 December, at 8am, online

2. Sample Exam Questions
   - limits on speedup
   - space time diagram
   - registers and shared memory
   - convolutions

MCS 572 Lecture 39
Introduction to Supercomputing
Jan Verschelde, 25 November 2024

Review for the Final Exam

1. The Final Exam
   - on Tuesday 10 December, at 8am, online

2. Sample Exam Questions
   - limits on speedup
   - space time diagram
   - registers and shared memory
   - convolutions

the final exam

- On Tuesday 10 December, at 8am, online.
- Runs in two versions:
  1. without computations, due by 10am, same day.
  2. with computations, due on Wednesday 11 December, at 9pm.
- We covered
  - parallel distributed memory computing,
  - parallel shared memory computing, and
  - GPU accelerated computing.

Review for the Final Exam

1. The Final Exam
   - on Tuesday 10 December, at 8am, online

2. Sample Exam Questions
   - limits on speedup
   - space time diagram
   - registers and shared memory
   - convolutions

limits on speedup

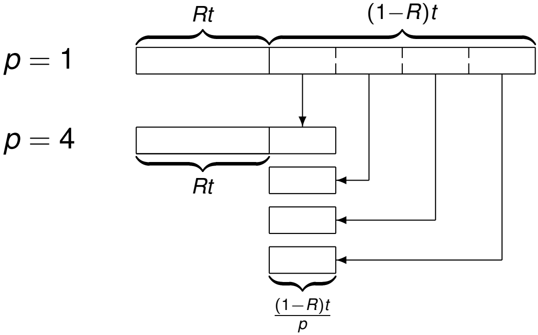
In a computation,
one fifth of all operations must be performed sequentially.

What is the maximum speedup that can be obtained
with ten processors?

Justify your answer.

applying Amdahl's Law

Consider a job that takes time t on one processor.
Let R be the fraction of t that must be done sequentially, $R = 0.2$.



Speedup on p processors $S(p) \leq \dfrac{t}{Rt + \frac{(1-R)t}{p}} = \dfrac{1}{R + \frac{1-R}{p}} \leq \dfrac{1}{R} = 5.$

$S(10) \leq 1/(1/5 + 8/100) = 100/28.$

Review for the Final Exam

1. The Final Exam
   - on Tuesday 10 December, at 8am, online

2. Sample Exam Questions
   - limits on speedup
   - space time diagram
   - registers and shared memory
   - convolutions

space time diagram

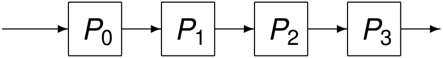
Consider a 4-stage pipeline, where each stage requires the same amount of processing time.

Draw the space time diagram to process 10 units.

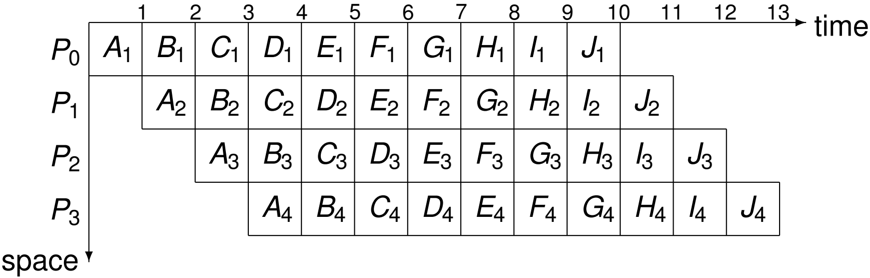
Use the diagram to justify the speedup.

space time diagram

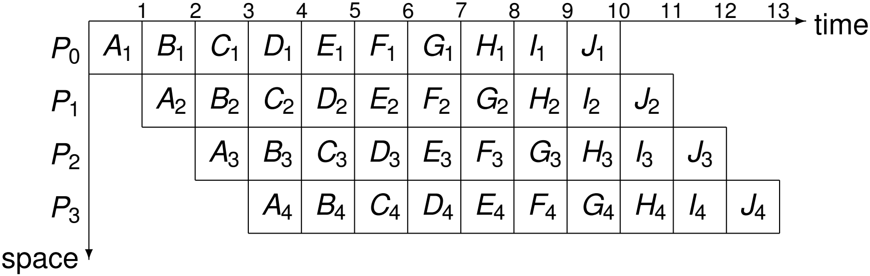
A pipeline of four processors P_0, P_1, P_2, P_3:



The space time diagram to process 10 elements:



the speedup



- Parallel time: 13.
- Sequential time: $10 \times 4 = 40$.

So, the speedup is $40/13 \approx 3.08$.

Review for the Final Exam

1. The Final Exam
   - on Tuesday 10 December, at 8am, online

2. Sample Exam Questions
   - limits on speedup
   - space time diagram
   - registers and shared memory
   - convolutions

registers and shared memory

Assume a CUDA kernel is launched with 8 thread blocks, with 512 threads in each block.

1. The kernel defines a local variable.
   How many copies of this variable exist during the execution of the kernel?

2. The kernel define a variable as shared memory.
   How many copies of this variable exist during the execution of the kernel?

Justify the number in your answer.

registers and shared memory

Assume a CUDA kernel is launched with 8 thread blocks, with 512 threads in each block.

1. The kernel defines a local variable.
   How many copies of this variable exist during the execution of the kernel?

Answer: 8×512 copies of the variable, because local variables are stored in registers, and each thread has its own registers.

registers and shared memory

Assume a CUDA kernel is launched with 8 thread blocks,
with 512 threads in each block.

2. The kernel define a variable as shared memory.
   How many copies of this variable exist
   during the execution of the kernel?

Answer: 8 copies of the variable,
because shared memory is shared between all threads in a block and
because there are 8 blocks, there are 8 copies of the variable.

Review for the Final Exam

1. The Final Exam
   - on Tuesday 10 December, at 8am, online

2. Sample Exam Questions
   - limits on speedup
   - space time diagram
   - registers and shared memory
   - convolutions

convolutions

Consider the convolution of two power series
given by coefficients in x and y:

$$z_k = \sum_{i=0}^{k} x_i y_{k-1},$$

where z_k is a coefficient of the series $z_0 + z_1 t + z_2 t^2 + \cdots$.
A basic implementation is given in the kernel below:

```
__global__ void convolute
( double *x, double *y, double *z )
{
   int k = threadIdx.x;      // thread k computes z[k]
   z[k] = x[0]*y[k];
   for(int i=1; i<=k; i++) z[k] = z[k] + x[i]*y[k-i];
}
```

CGMA and thread divergence

The kernel on the previous slide is called for one block of threads.

You may assume that the number of threads in the block equals the dimension of the arrays x, y, and z.

1. What is the Compute to Global Memory Access ratio for the kernel?
2. Change the kernel into an equivalent one using fewer global memory accesses.
3. What is the CGMA ratio for your new kernel?
4. Explain the thread divergence of the given kernel. Describe a way to eliminate the thread divergence.

Compute to Global Memory Access ratio

```
__global__ void convolute
( double *x, double *y, double *z )
{
   int k = threadIdx.x;     // thread k computes z[k]
   z[k] = x[0]*y[k];
   for(int i=1; i<=k; i++) z[k] = z[k] + x[i]*y[k-i];
}
```

- `z[k] = x[0]*y[k]` does one multiplication and 3 memory accesses;
- `z[k] = z[k] + x[i]*y[k-i]` does one addition, one multiplication, and 4 memory accesses.

Therefore, the CGMA ratio is $\frac{1}{3} + \frac{2\mathtt{k}}{4\mathtt{k}}$ for thread $\mathtt{k}$.

a new kernel with fewer memory accesses

```
__global__ void convolute
( double *x, double *y, double *z )
{
   int k = threadIdx.x;    // thread k computes z[k]
   double zk;              // a register stores z[k]
   __shared__ xv[MAX];     // store x and y
   __shared__ yv[MAX];     // into shared memory
   xv[k] = x[k];           // assume dimension
   yv[k] = y[k];           // equals #threads
   __syncthreads;
   zk = xv[0]*yv[k];
   for(int i=1; i<=k; i++) zk = zk + xv[i]*yv[k-i];
   z[k] = zk;
}
```

The CGMA ratio is $\frac{1+2k}{3}$.

thread divergence

```
__global__ void convolute
( double *x, double *y, double *z )
{
   int k = threadIdx.x;     // thread k computes z[k]
   z[k] = x[0]*y[k];
   for(int i=1; i<=k; i++) z[k] = z[k] + x[i]*y[k-i];
}
```

Because of the test `i<=k`,
each thread does a different number of operations.

Because all threads in a warp execute the same instruction,
executing this kernel will require twice the `blockDim.x`.

zero padding eliminates thread divergence

```
__global__ void convolute
( double *x, double *y, double *z )
{
   int k = threadIdx.x;    // thread k computes z[k]
   int dim = blockDim.x;
   double zk;              // a register stores z[k]
   __shared__ xv[MAX];     // store x and y
   __shared__ yv[2*MAX];   // into shared memory
   xv[k] = x[k];           // assume dimension
   yv[k] = 0.0;
   yv[k] = y[dim+k];       // equals #threads
   __syncthreads;
   zk = xv[0]*yv[dim+k];
   for(int i=1; i<=dim; i++)
      zk = zk + xv[i]*yv[k+dim-i];
   z[k] = zk;
}
```