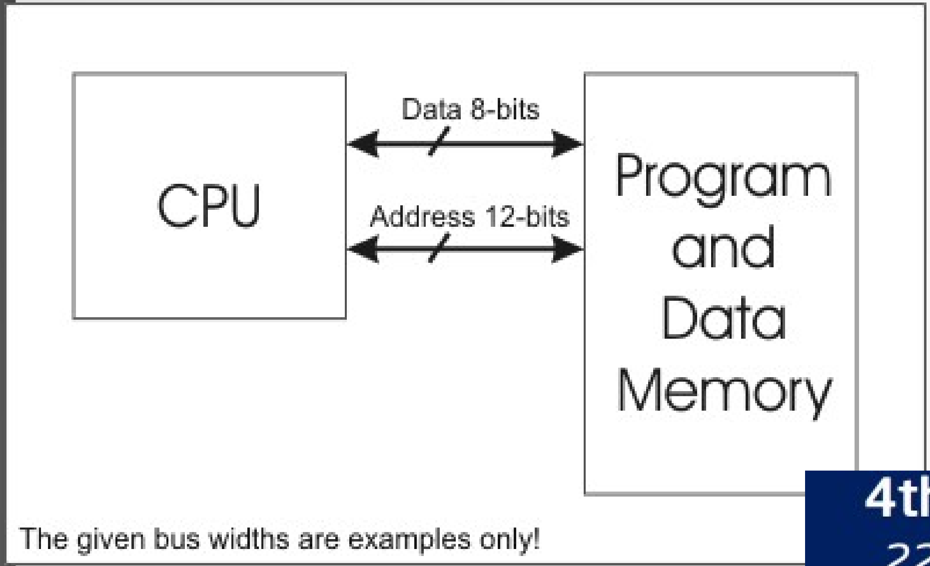
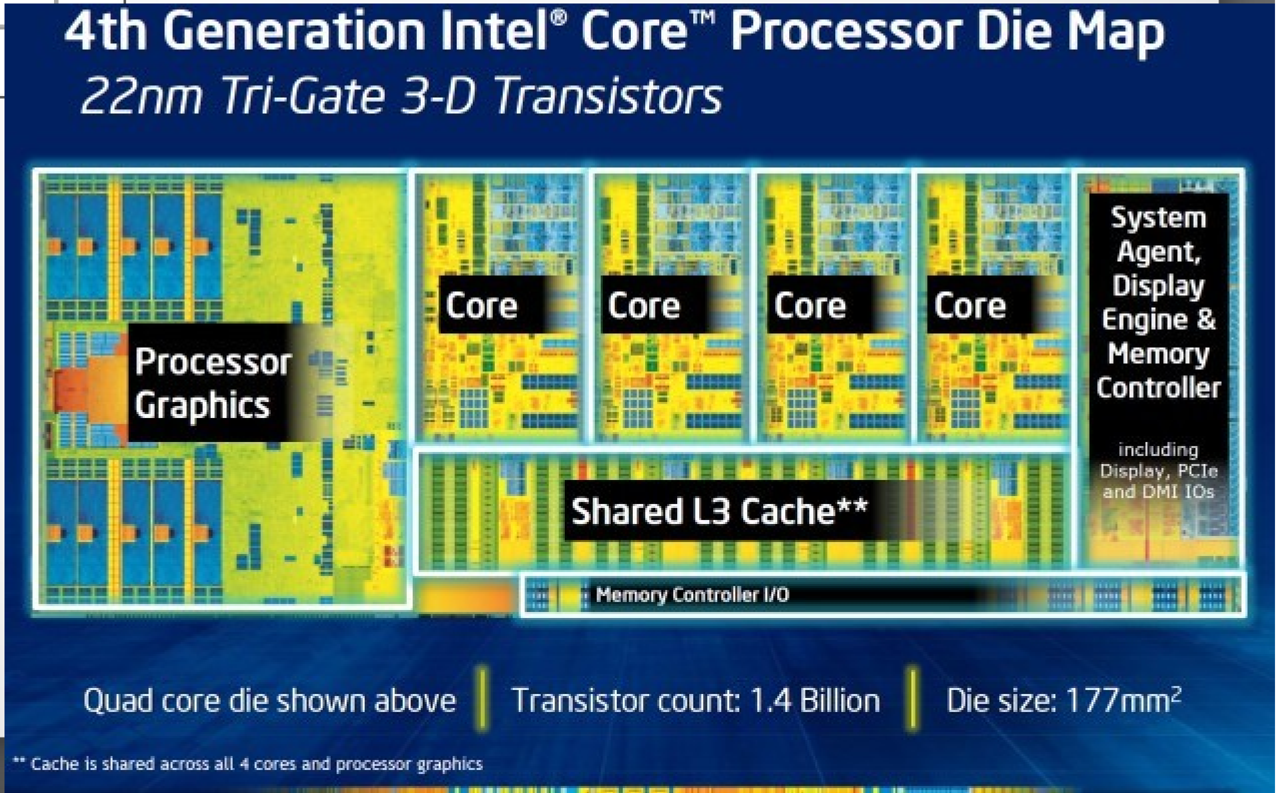
Memory models and compiler optimisation

Krishna Kumar

- Memory model we like to program for



How it actually looks



Key Assumption:

Maintain fully coherent caches

Does your compiler execute the program you wrote?

- Sequential consistency: Executing the program you wrote.
- “the result of any execution is the same as if the operations of all the processors were executed in some sequential order, and **the operations of each individual processor appear in this sequence in the order specified by its program.**” - Lesslie Lamport
- Compiler optimisation
- Processor execution
- Cache coherency
- Chip / compiler design annoyingly helpful:
  - It can be expensive to exactly execute what you wrote
  - Often they rather do something else, that's faster.

Transformation of code

Source

Compiler

Processor

Cache

Execution

Compiler optimisations

```
for (i = 0; i < rows; ++i) {
    for (j = 0; j < cols; ++j){
        a[j*rows+i]+=42;
    }
}
```

```
for (j = 0; j < cols; ++j) {
    for (i = 0; j < rows; ++i)
    {
        a[j*rows+i]+=42;
    }
}
```

Dekker's and Peterson's Algorithm

Consider (flags are shared and atomic *but unordered*, initially set to zero.)

- Thread 1:

```
flag1 = 1; //a: declare intent
if (flag2 != 0) // b
    // resolve contention
else
    // enter critical section
```

- Thread 2:

```
flag2 = 1; //c: declare intent
if (flag1 != 0) // d
    // resolve contention
else
    // enter critical section
```

If a can pass b, or c can pass d, this breaks!

Store Buffer

Processor 1

```
flag 1 = 1; // a
if (flag2 != 0)  // b
{
   ...
}
```

store buffer

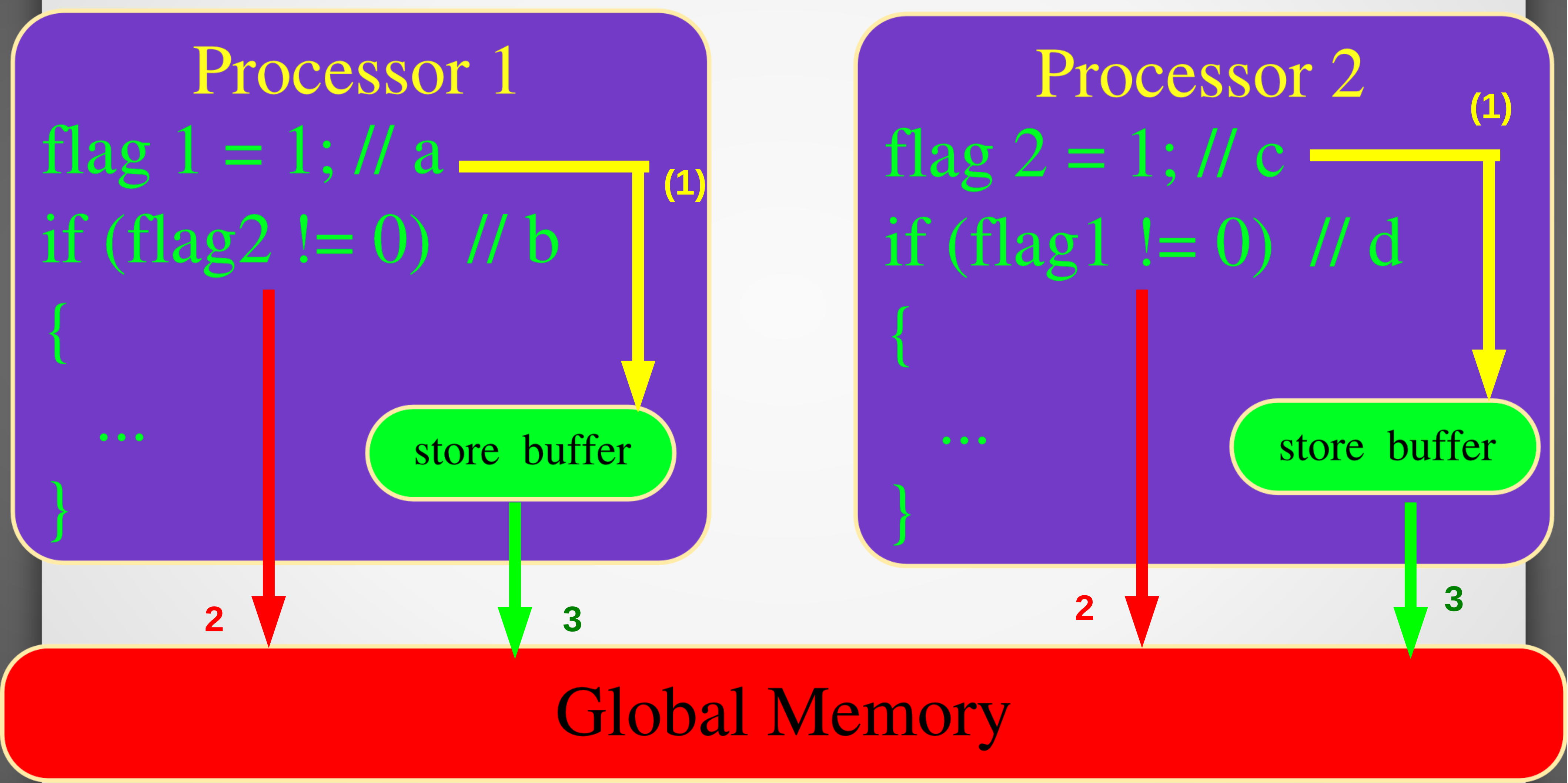
Processor 2

```
flag 2 = 1; // c
if (flag1  != 0)  // d
{
   ...
}
```

store buffer

Global Memory

Store Buffer



Compiler Optimisations (cont...)

- X = 1;
- Y = 2;
- Z = 3;// using X, Y, Z

- Z = 3;
- Y = 2;
- X = 1; // using X, Y, Z

- X = 1;
- Y = 2;
- X = 3;// using X & Y

- Y = 2;
- X = 3;// using X & Y

POSIX - Threads

- Portable Operating System Interface (POSIX) Threads, or Pthreads, is a POSIX standard for threads.
- The POSIX thread libraries are a standards based thread API for C/C++
- **Thread management:**
  - Routines that work directly on threads - creating, detaching, joining, etc.
- **Mutexes:**
  - Mutex functions provide for creating, destroying, locking and unlocking mutexes.
- **Condition variables:**
  - Routines that address communications between threads that share a mutex.
- **Synchronization:**
  - Routines that manage read/write locks and barriers.

POSIX - Pthreads

- g++ -lpthread -fpermissive
- **pthread_create (pthread_t *thread, pthread_attr_t *attr, void *(*start_routine)(void *), void *arg) :**

- **Thread:** An identifier for the new thread returned by the subroutine. This is a pointer to pthread_t structure. .
- **attr:** An attribute object that may be used to set thread attributes. We can specify a thread attributes object, or NULL for the default values.
- **start_routine:**
  - The routine that the thread will execute once it is created.
  - void *(*start_routine)(void *) We should pass the address of a function taking a pointer to void as a parameter and the function will return a pointer to void. So, we can pass any type of single argument and return a pointer to any type.
- **arg:**
  - A single argument that may be passed to start_routine. It must be passed as a void pointer. NULL may be used if no argument is to be passed.

POSIX – Files and Directories <unistd.h>

- **char *getcwd(char *buf, size_t size);** - get current working directory
- **int mkdir(const char *pathname, mode_t mode);** - create a directory
- **int rmdir(const char *pathname);** - delete a directory
- **int chdir(const char *path);** change working directory
- **int link(const char *oldpath, const char *newpath);** make a new name for a file
- **int unlink(const char *pathname);** delete a name and possibly the file it refers to
- **int rename(const char *oldpath, const char *newpath);** change the name or location of a file
- **int stat(const char *file_name, struct stat *buf);** get file status
- **int chmod(const char *path, mode_t mode);** change permissions of a file
- **int chown(const char *path, uid_t owner, gid_t group);** change ownership of a file
- **DIR *opendir(const char *name);** open a directory
- **struct dirent *readdir(DIR *dir);** read directory entry
- **int closedir(DIR *dir);** close a directory

More POSIX

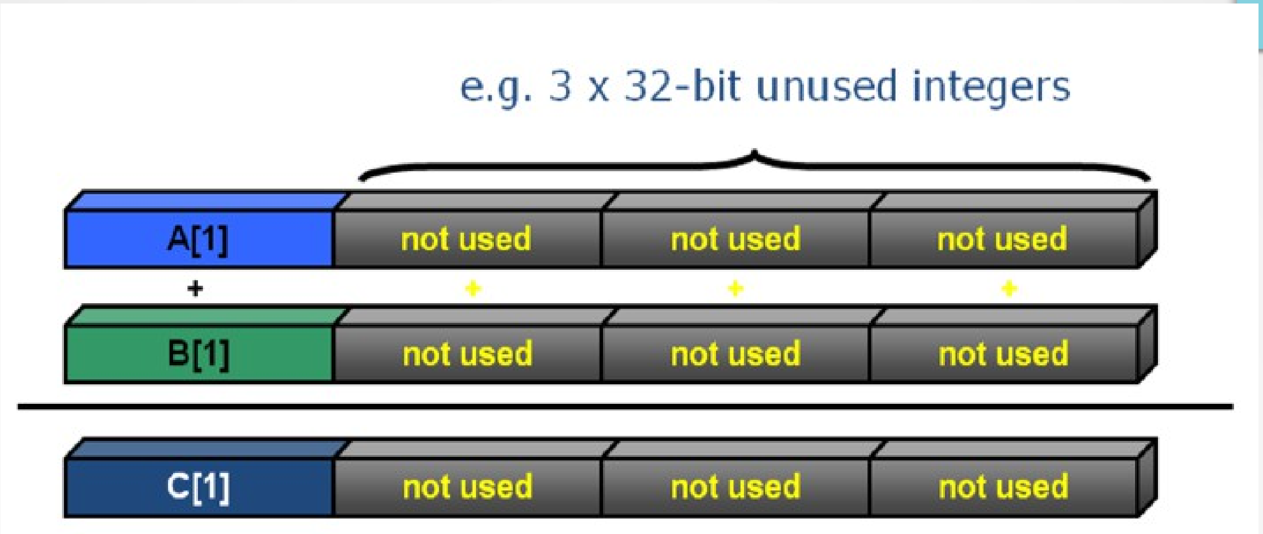
- Advanced File Operations
- Processes
- Long Jumps - #include <setjmp.h> - Stacks
- Signal Handling - #include <signal.h>
- Obtaining Information at Runtime
- Terminal I/O
- Process Groups and Job Control

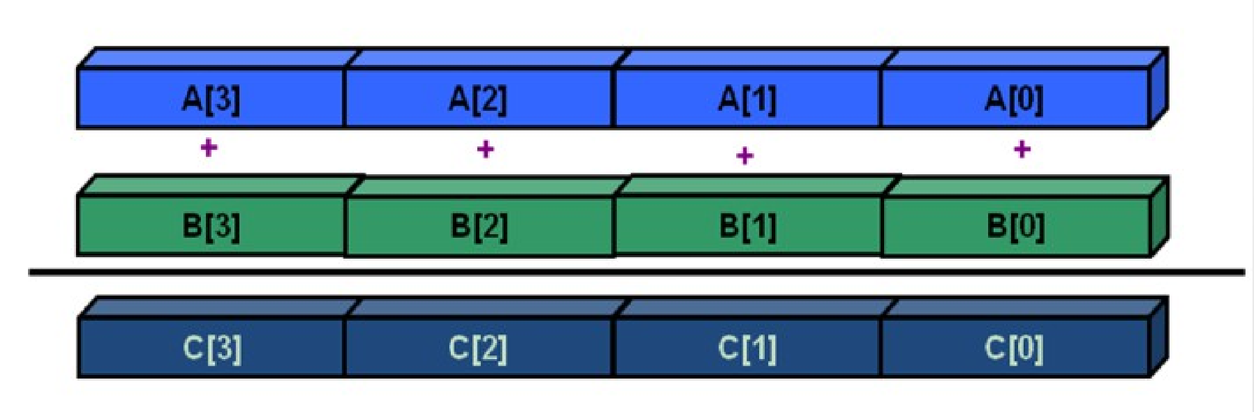
Auto Vectorization

- for (int i = 0; i < 1000; ++i)
  - A[i] = B[i]*C[i]; //32-bit operation
- for (int i = 0; i < 1000; i+=4)
  - A[i:i+3] = mulps(B[i:i+3]*C[i:i+3]); // 128-bit
  - Which is 4x32bit operations
  - Takes the same amount of time

The Auto-Vectorizer analyzes loops in your code, and uses the vector registers and instructions on the target computer to execute them, if it can. This can improve the performance of your code.

How auto-vectorization works?





SIMD Parallelism in Loops

- Loop level parallelism
  - SIMD for a single statement
  - across consecutive iterations
- Handles
  - Misaligned data
  - Patterns such as reduction, linear recursion
  - Employ cost models to amortize overhead in versionin alignment

SIMD Parallelism in Basic Blocks

```
for (i=0; i<N; i+=4) {
    a[i] = b[i] + c[i];
    a[i+1] = b[i+1] + c[i+1];
    a[i+2] = b[i+2] + c[i+2];
    a[i+3] = b[i+3] + c[i+3];
}
```

unrolled loop

```
for (i=0; i<N; i+=4) {
    a[i:i+3] = b[i:i+3] + c[i:i+3];
}
```

```
for (i=0; i<N; i++) {
    p = &a[i]; q = &b[i];
    p.x = q.x + …
    p.y = q.y + …
    p.z = q.z + …
}
```

structure

```
for (i=0; i<N; i++) {
    p = &a[i]; q = &b[i];
    p.xyz = q.xyz + …
}
```

x	y	z	*dummy*

```
s += a[i]*b[i] + a[i+1]*b[i+1] +
  a[i+2]*b[i+2] + a[i+3]*b[i+3];
```

statement

```
t[i:i+3] = a[i:i+3] * b[i:i+3];
s += t[i]+t[i+1]+t[i+2]+t[i+3];
```

SIMD Parallelism in Short Loops

- SIMD across entire loop iterations
- Effectively collapse innermost loop
- Allow to extract SIMD at the next loop level

```
for (k=0; k<N; k++) {

   ...other code...

   for (i=0; i<8; i++)
        r[k] += i[k+i] * c[k+i];
}
```

```
for (k=0; k<N; k++) {

   ...other code...

   r[k:k+3] = i[k+i:k+3+i]*c[k+i:k+3+i];
   r[k+4:k+7] = i[k+4+i:k+7+i]*c[k+4+i:k+7+i];
}
```

References

- http://arcs.skku.edu/pmwiki/uploads/Courses/MulticoreSy ems/05-SIMD.pdf
- https://d3f8ykwhia686p.cloudfront.net/1live/intel/Compile utovectorizationGuide.pdf
- http://cplus.kompf.de/posixlist.html#i1647905531
- http://arcs.skku.edu/pmwiki/uploads/Courses/MulticoreSy ems/05-SIMD.pdf