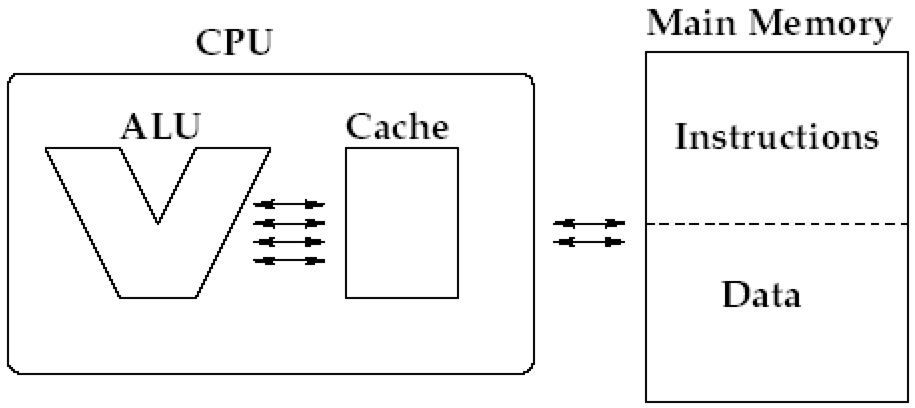
Stream Processing/ Computing

Computing Architectures

- Von Neumann
  - traditional CPUs
- Systolic arrays
- SIMD architectures
  - for example, Intel’s SSE, MMX
- Vector processors
- Stream processors
  - GPUs are a form of these

Von Neumann



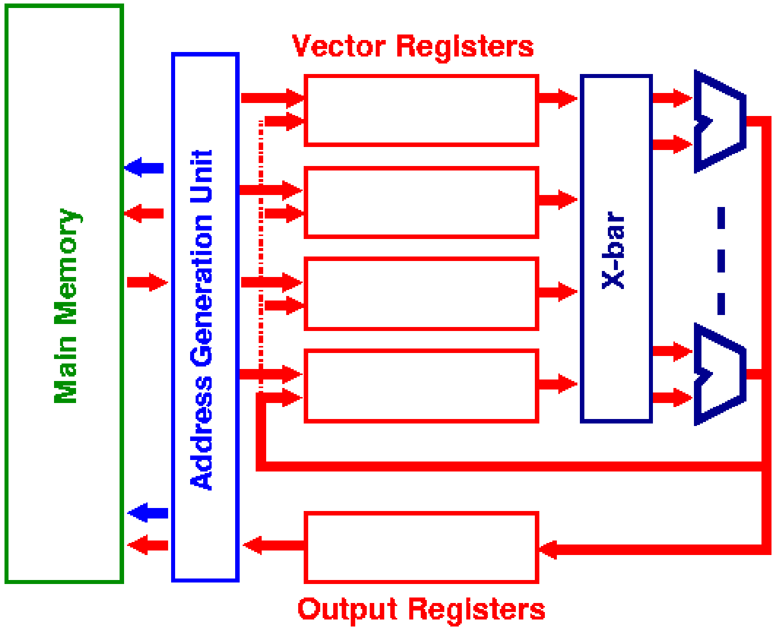
- Classic form of programmable processor
- Creates Von-Neumann bottleneck
  - separation of memory and ALU creates bandwidth problems
  - today’s ALUs are much faster than today’s data links
  - this limits compute-intensive applications
  - cache management to overcome slow data links adds to control overhead

Early Streamers

- Systolic Arrays
  - arrange computational units in a specific topology (ring, line)
  - data flow from one unit to the next
- SIMD (Same Instruction Multiple Data)
  - a single set of instructions is executed by different processors in a collection
  - multiple data streams are presented to each processing unit
  - SSE, MMX is a 4-way SIMD, but still requires instruction decode for each word

Early Streamers

- Vector processors
  - made popular by Cray supercomputers
  - represent data as a vector
  - load vector with a single instruction (amortizes instruction decode overhead)
  - exposes data parallelism by operating on large aggregates of data



Stream Processors – Motivation

- In VLSI technology, computing is cheap
  - thousands of arithmetic logic units operating at multiple GHz can fit on 1cm^2 die
- But… delivering instructions and data to these is expensive
- Example:
  - only 6.5% of the Itanium die is devoted to its 12 integer and 2 floating point ALUs and their registers
  - remainder is used for communication, control, and storage

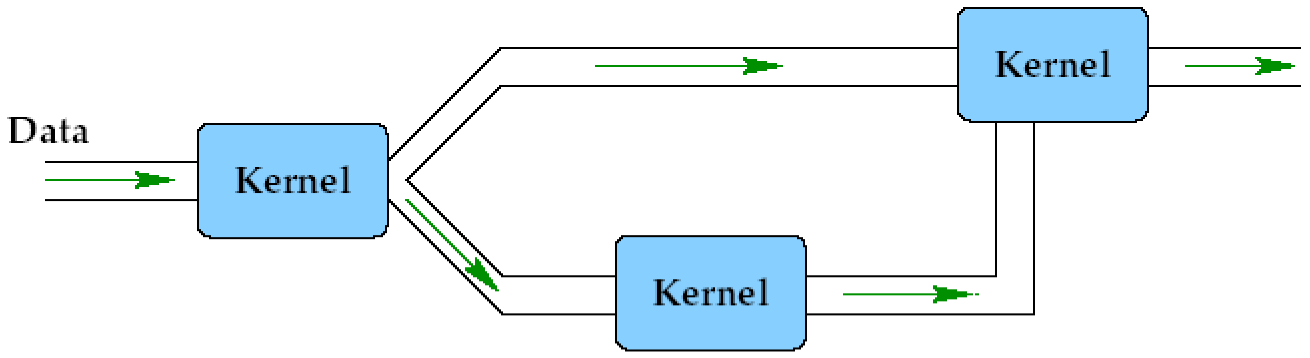
Stream Processors – Motivation

- Thus, general-purpose use of CPUs comes at a price
- In contrast:
  - the more efficient control and communication on the Nvidia GeForce4 enables the use of many hundreds of floating-point and integer ALUs
  - for the special purpose of rendering 3D images
  - this task exposes abundant parallelism
  - requires little global communication and storage

Stream Processors – Motivation

- Goal:
  - expose these patterns of communication, control, and parallelism to a wider class of applications
  - Create a general purpose streaming architecture without compromising its advantages
- Proposed streaming architectures
  - Imagine (Stanford)
  - CHEOPS
- Existing implementations that come close
  - Nvidia FX, ATI Radeon GPUs
  - enable GP-GPU (general purpose streaming, GP-GPU)

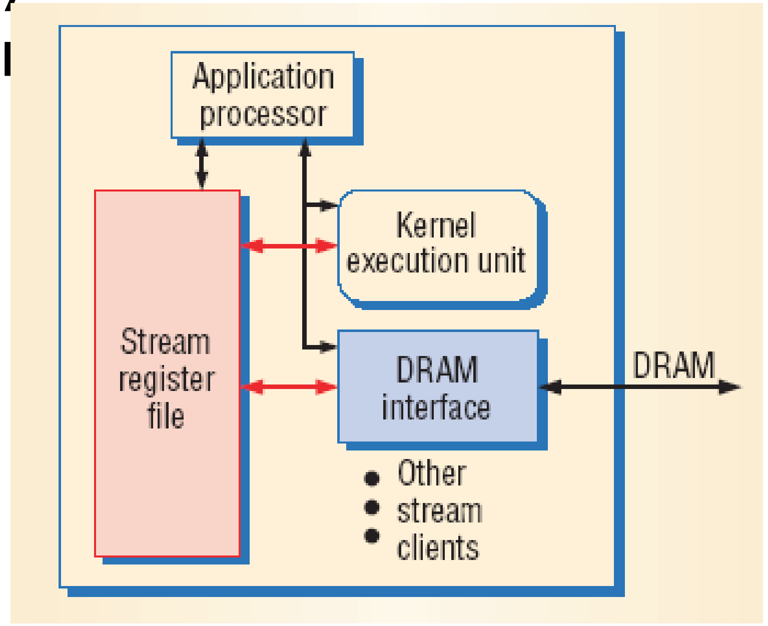
Stream Processing



- Organize an application into streams and kernels
    - expose inherent locality and concurrency (here, of media-processing applications)
- This creates the programming model for stream processors
    - and therefore also for GPGPU

Stream Proc. – Memory Hierarchy

- Local register files (LRFs)
  - operands for arithmetic operations (similar to caches on CPUs)
  - exploit fine-grain locality
- Stream register files (SRFs)
  - capture coarse-grain locality
  - efficiently transfer data to a
    from the LRFs
- Off-chip memory
  - store global data
  - only use when necessary



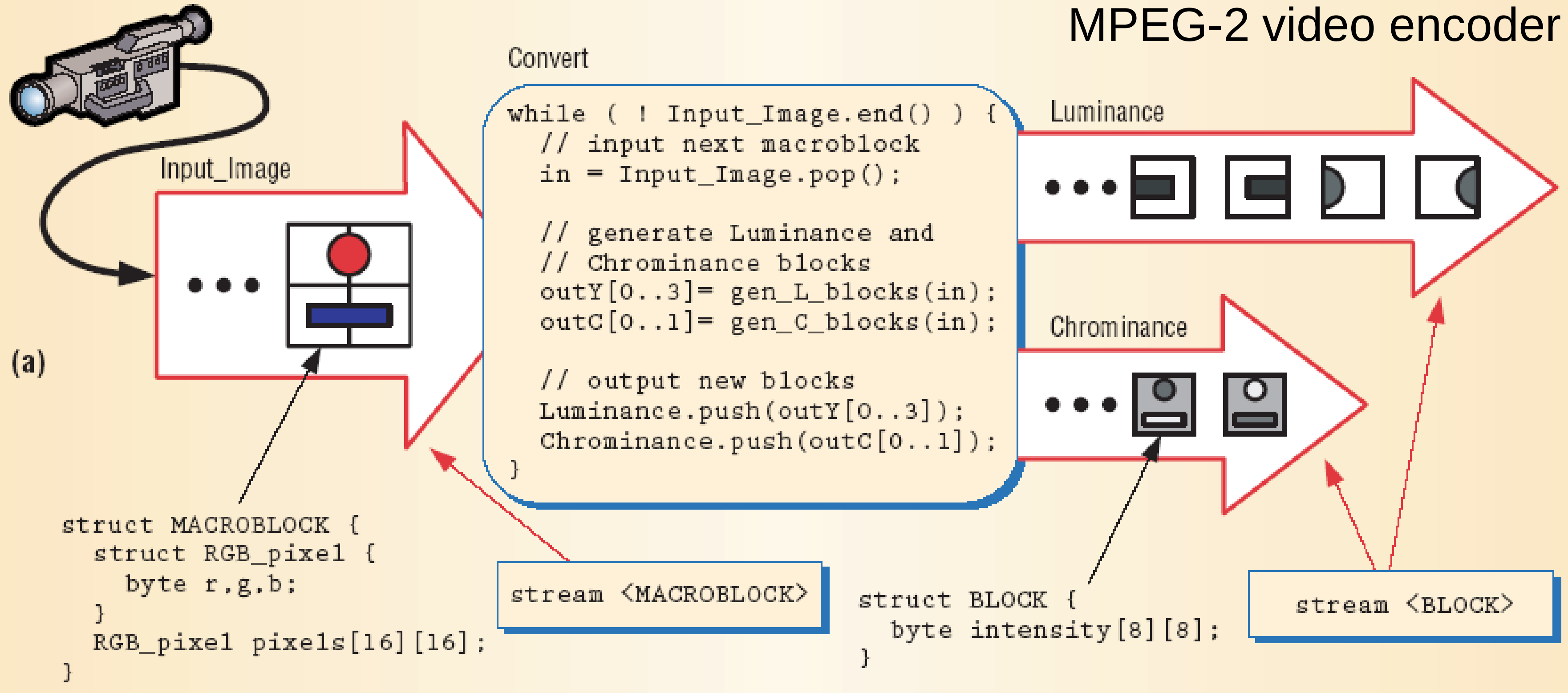
Stream Proc. – Memory Hierarchy

- These form a bandwidth hierarchy as well
  - roughly an order of magnitude for each level
  - well matched by today's VLSI technology
- By exploiting the locality of media operations the hundreds of ALUs can operate at peak rate
- While CPUs and DSPs rely on global storage and communication, stream processors get more "bang" out of a die

Stream Processing – Example 1

MPEG-2 video encoder

(a)



```
while ( ! Input_Image.end() ) {
  // input next macroblock
  in = Input_Image.pop();

  // generate Luminance and
  // Chrominance blocks
  outY[0..3]= gen_L_blocks(in);
  outC[0..1]= gen_C_blocks(in);

  // output new blocks
  Luminance.push(outY[0..3]);
  Chrominance.push(outC[0..1]);
}
```

```
struct MACROBLOCK {
  struct RGB_pixel {
    byte r,g,b;
  }
  RGB_pixel pixels[16][16];
}
```

```
stream <MACROBLOCK>
```

```
struct BLOCK {
  byte intensity[8][8];
}
```

```
stream <BLOCK>
```

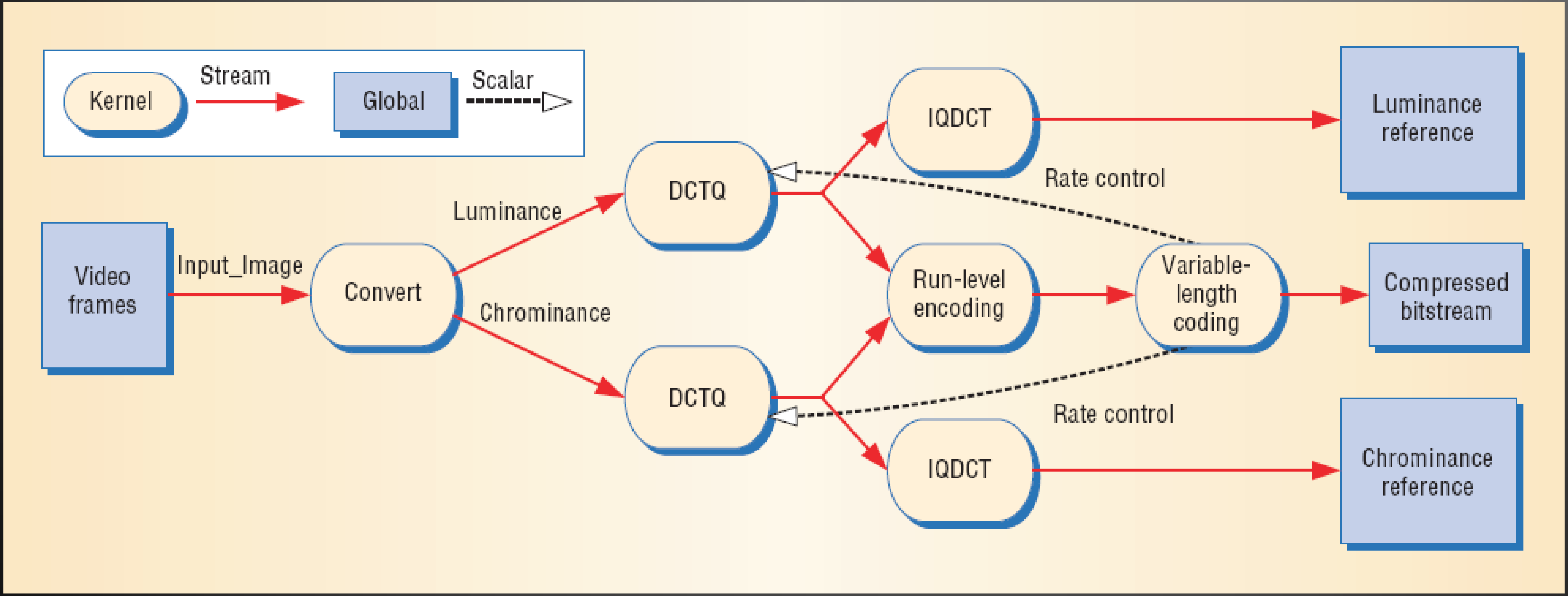
(b)

```
stream <MACROBLOCK> Input_Image(NUM_MB);
stream <BLOCK> Luminance(NUM_MB*4), Chrominance(NUM_MB*2),

Input_Image = Video_Feed.get_macroblocks(currpos, NUM_MB);
currpos += NUM_MB;
Convert(Input_Image, Luminance, Chrominance);
```

Stream-C program

Stream Processing – Example 2



MPEG-2 I-frame encoder

Q: Quantization, IQ: Inverse Quantization, DCT: Discrete Cosine Transform

Global communication (from RAM) needed for the reference frames (needed to ensure persistent information)

Stream Processing – Parallelism

- Instruction-level
  - exploit parallelism in the scalar operations within a kernel
  - for example, gen_L_blocks, gen_C_blocks can occur in parallel
- Data-level
  - operate on several data items within a stream in parallel
  - for example, different blocks can be converted simultaneously
  - note, however, that this gives up the benefits that come with sequential processing (see later)
- Task parallelism
  - must obey dependencies in the stream graph
  - for example, the two DCTQ kernels could run in parallel

Stream Processors vs. GPUs

- Stream elements
  - points (vertex stage)
  - fragments, essentially pixels (fragment stage)
- Kernels
  - vertex and fragment shaders
- Memory
  - texture memory (SRFs)
  - not-exposed LRF, if at all
  - bandwidth to RAM better, with PCI-Express

Stream Processors vs. GPUs

- Data parallelism
  - fragments and points are processed in parallel
- Task parallelism
  - fragment and vertex shaders can work in parallel
  - data trabsfer from RAM can be overlapped with computation
- Instruction parallelism
  - see task-parallelism

Stream Processors vs. GPUs

- Stream processors allow looping and jumping
  - not possible on GPUs (at least not straighforward)
- Stream processors follow a Turing machine
  - GPUs are restricted (see above)
- Stream processors have much more memory
  - GPUs have 256 MB, soon 512 MB

Conclusions

- GPUs are not 100% stream processors, but they come close
  - and one can actually buy them, cheaply
- Loss of jumps and loops enforces pipeline discipline
- Lack of memory allows use of small caches and prevents swamping the chip with data
- Data parallism often requires task decomposition into multiple passes (see later)

References

- U. Kapasi, S. Rixner, W. Dally et al. "Programmable stream processors," IEEE Computer August 2003
- S.Venkatasubramanian, "The graphics card as a stream computer," SIGMOD DIMACS, 2003