9 Data Level Parallelism

Data Level Parallelism can be executed in ILP architecture, but there is a better solution.

- **Data Level Parallelism** is more explicit;
- **Instruction Level Parallelism** is more implicit, and it need ILP architecture to find out and speed up.

SISD (Single Instruction Single Data), the processor can execute an instruction operating on single data, such as the basic processor we have learnt before.

SIMD (Single Instruction Multiple Data): the processor can execute an instruction operating on multiple data (such as vector), such as vector machine.

Vector Machine （向量机） can operate to a vector of data in the same time; comparing to scalar machine（标量机）: less general but more efficient.

- Example: vector multiplication. Do not have multiple multiplier; single multiplier but highly-pipelined.
- Vector chaining (bypassing).
- Masked vector instruction: add a **mask bit** （掩码） in every place of the vector.
  - If mask bit is 1, the place does not need to be operated;
  - If mask bit is 0, the place needs to be operated.

The Vector Extension of General Processor（通用处理器的向量化扩展）(SIMD-extension)

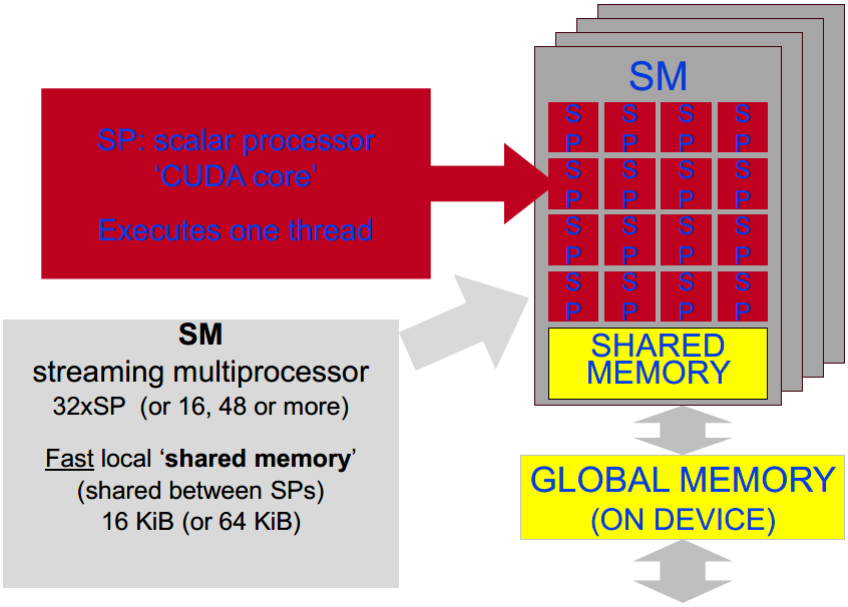
- Multimedia Extensions (SIMD extensions)
  - Add some small-length vector operations.
    - A 64 bits register decomposed to 2×32 bits, or 4×16 bits, or 8×8 bits.

      (For example, RGB codes only need 8 bit)
  - Special design some larger registers (128 bits, 256 bits, 512 bits)
  - Add some SIMD instructions.
- No $vlen$ value (vector length), and the user should specify the length of the vector (only 1, 2, 4, 8). For example, `ADD.4D F8 F8 F4` (F8 and F4 are regarded as 4-length vectors).
- SIMD CPU Example: Intel Core i7. 4 cores, 8 SIMD ALUs per core (TLP, DLP supported).

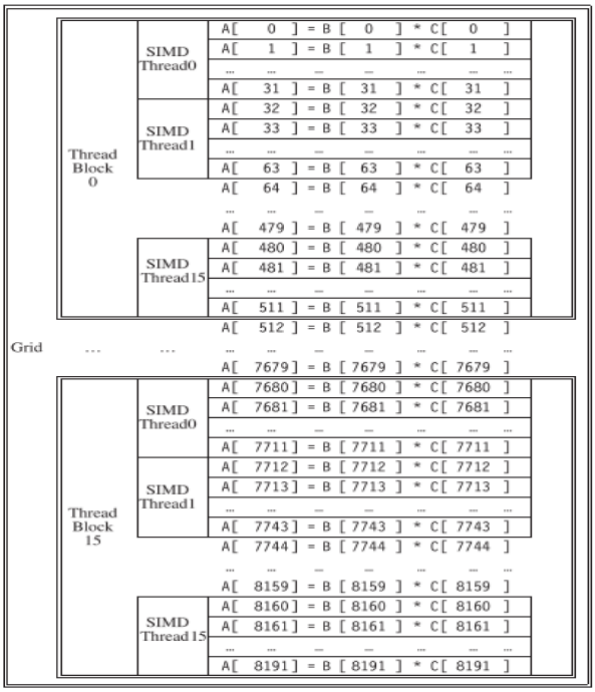
SISD, SIMD, MISD, MIMD

- Single Instruction Stream Single Data Stream **(SISD)**;
  - Traditional superscalar is SISD system (One instruction stream, one data stream, only allow parallel execution).
- Single Instruction Stream Multiple Data Stream **(SIMD)**: introduced above.
- Multiple Instruction Stream Single Data Stream **(MISD)**: rarely seen, used in fault-tolerance system;
- Multiple Instruction Stream Multiple Data Stream **(MIMD)**: multiple SISD system (multi-core, multi-processor, etc)

GPU (Graphic Processing Unit)



- Support both TLP and DLP;
  - DLP: multiple scalar processor 'CUDA core' in one streaming multiprocessor;
  - TLP: multiple streaming multiprocessor (SM).
- The calculation is done by independent CUDA threads (micro-threads), combine these threads together to form a thread block.
- CUDA cores (micro-cores): SIMD (Single Instruction Multiple Data); cores (parallel core): MIMD (Multiple Instruction Multiple Data)
- **Single Program/procedure Multiple Data (SIPD)**
  - The same program, different data stream, can execute different operations (in the same program, by checking the id and so on).
  - Parallel Programming.
- GPU use SIPD programming instead of SIMD programming.
- **Hardware support**:
  - **Thread block scheduler**: map the grid of thread blocks (a block contains several threads) into several different cores.
  - **SIMD thread scheduler**: schedule the threads in a thread block in several micro-cores in one GPU core.



- Combine several threads together and get a **warp** （束）(aka SIMD thread).
- Every GPU core has several SIMD **lane**（车道）(also micro-core). Then regard a warp as an instruction and put the warp into the lanes to execute.

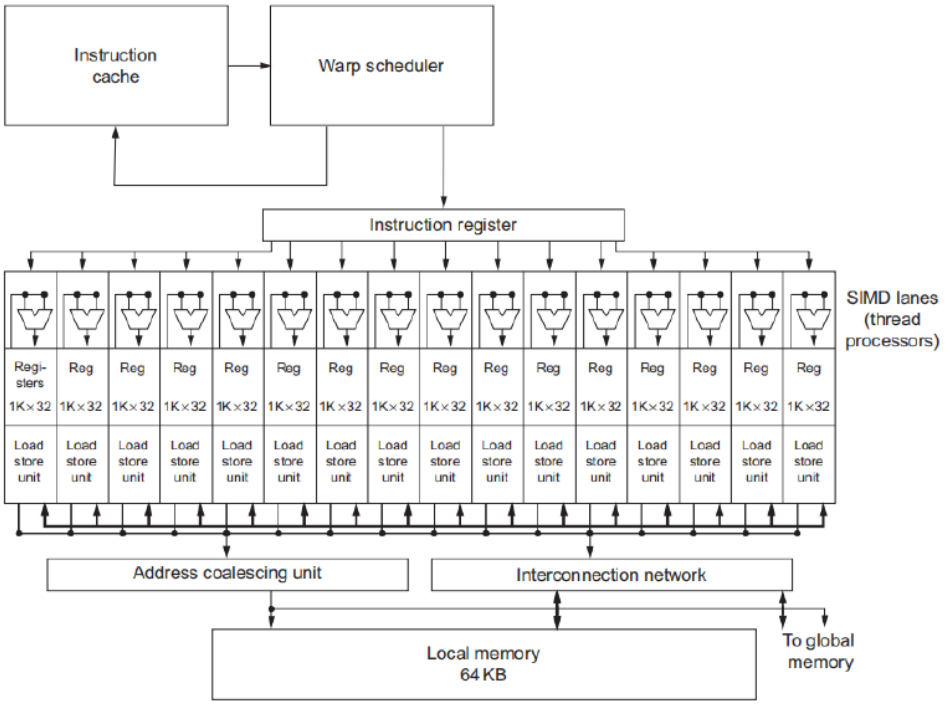
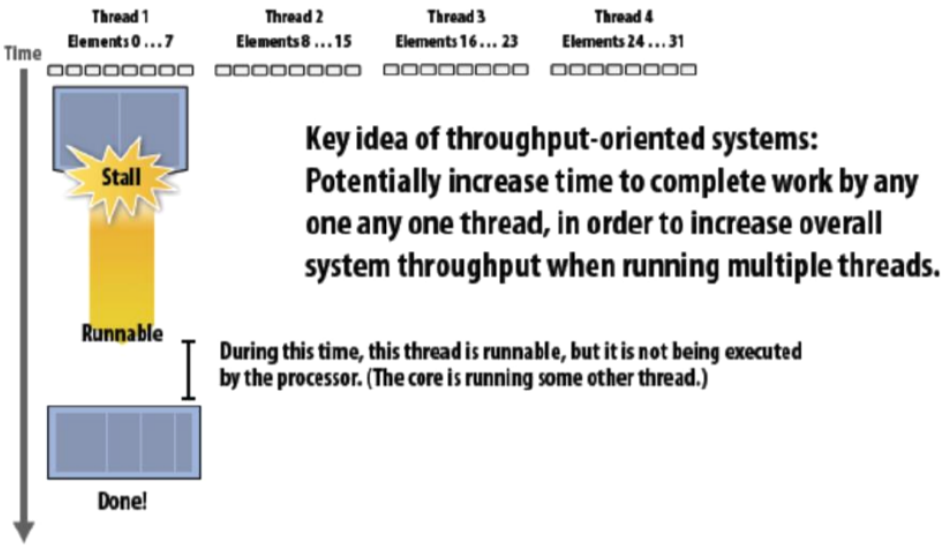


Figure 4.14 Simplified block diagram of a multithreaded SIMD Processor. It has 16 SIMD Lanes. The SIMD Thread Scheduler has, say, 64 independent threads of SIMD instructions that it schedules with a table of 64 program counters (PCs). Note that each lane has 1024 32-bit registers.

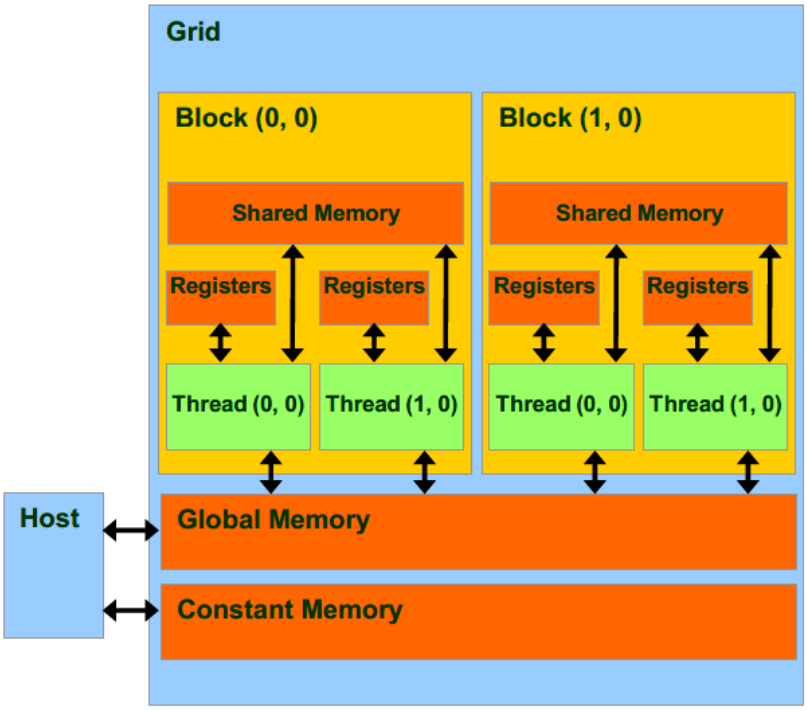
- Set **mask bit** to block some ALUs' output. Branches will slow down the speed of lanes because all the lanes will execute all the instructions, except some instructions we use mask bit to ignore the useless result.
- **SIMD thread scheduler** every time issue a ready instruction from a different SIMD thread, to hide the stall（隐藏延迟）. This method is called ***wrap-level FGMT*** (every time execute one instruction from one SIMD thread, no interlocking). Increase throughput, hide latency, but also increase the execution time of every thread.



- **Three Key Idea of GPU**
  - Employ multiple processing cores (simple core, focus on TLP);
  - Pack cores full of ALU (SIMD, focus on DLP);
  - Use multithread to improve the system throughput; execute different instructions from different in turn to hide latency.

Memory Model of CUDA and cooperating threads (G80 implementation)

- Each thread can:
  - read/write per-thread registers;
  - read/write per-thread local memory;
  - read/write per-block shared memory;
  - read/write per-grid global memory;
  - read per-grid constant memory.



- Use `__syncthreads()` as a barrier to prevent data hazards.