Tutorial 4: Compilers, linkers and performance

Informatik elective: GPU Computing

Pratik Nayak

NVIDIA GPU architectures

Arch →	Tesla	Fermi	Kepler	Maxwell	Pascal	Volta	Turing	Ampere	Ada Lovelace	Hopper	Blackwell
Compute capability →	1.0	2.0	3.0	5.x	6.x	7.0	7.5	8.0	8.9	9.0	Blackwell
GPUs →					P100	V100		A100		H100	

Summit supercomputer (ORNL, USA)

JUPITER (Jülich, DE)

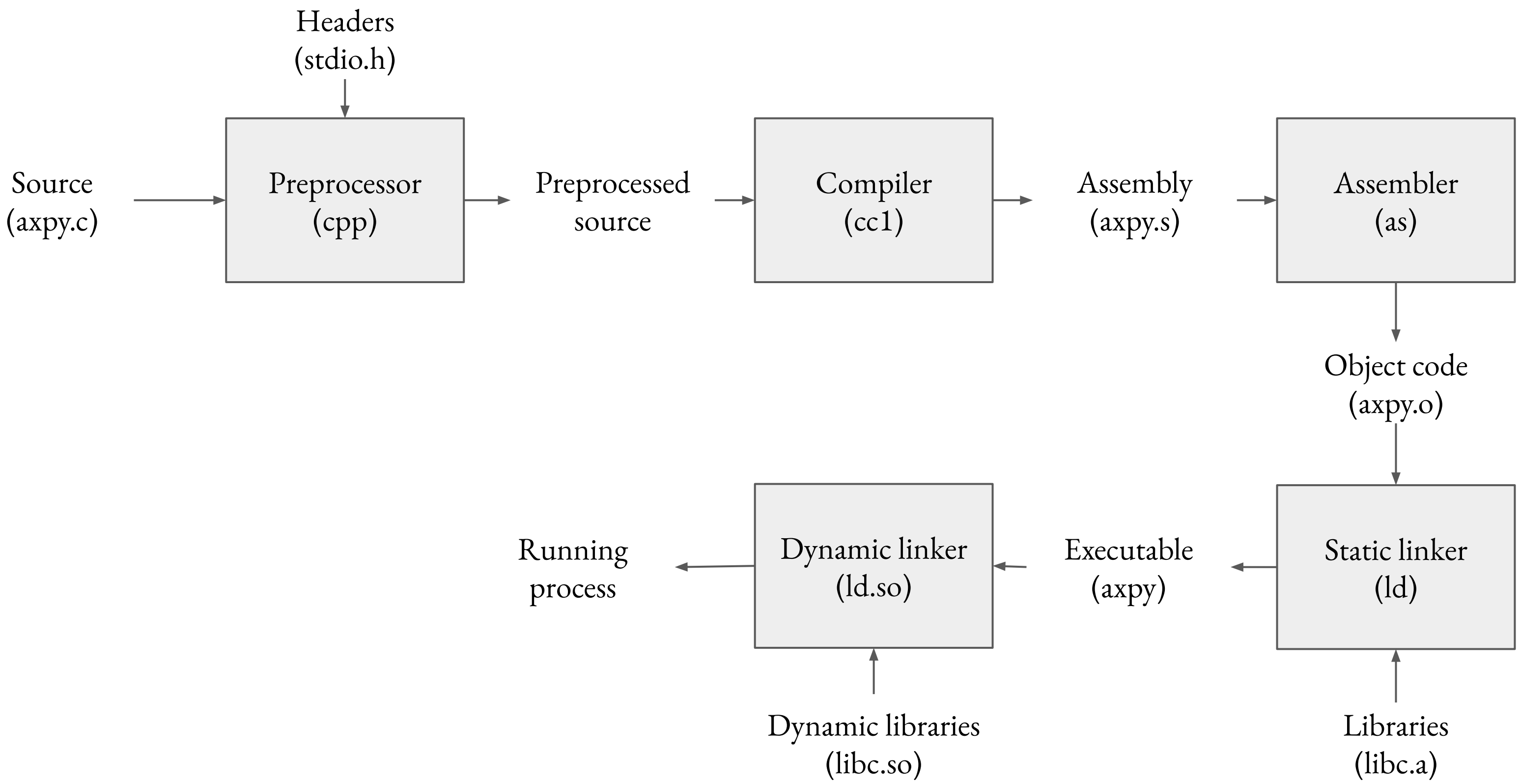
New architectures → new features, more performance.

NVIDIA GPU architectures

<table>
<tr><th rowspan="2">Feature support (unlisted features are supported for all compute capabilities)</th><th colspan="9">Compute capability (version)</th></tr>
<tr><th>1.0, 1.1</th><th>1.2, 1.3</th><th>2.x</th><th>3.0</th><th>3.2</th><th>3.5, 3.7, 5.x, 6.x, 7.0, 7.2</th><th>7.5</th><th>8.x</th><th>9.0</th></tr>
<tr><td>Warp vote functions (__all(), __any())</td><td>No</td><td colspan="8">Yes</td></tr>
<tr><td>Warp vote functions (__ballot())</td><td colspan="2" rowspan="5">No</td><td colspan="7" rowspan="5">Yes</td></tr>
<tr><td>Memory fence functions (__threadfence_system())</td></tr>
<tr><td>Synchronization functions (__syncthreads_count(), __syncthreads_and(), __syncthreads_or())</td></tr>
<tr><td>Surface functions</td></tr>
<tr><td>3D grid of thread blocks</td></tr>
<tr><td>Warp shuffle functions</td><td colspan="3" rowspan="2">No</td><td colspan="6" rowspan="2">Yes</td></tr>
<tr><td>Unified memory programming</td></tr>
<tr><td>Funnel shift</td><td colspan="4">No</td><td colspan="5">Yes</td></tr>
<tr><td>Dynamic parallelism</td><td colspan="5">No</td><td colspan="4">Yes</td></tr>
<tr><td>Uniform Datapath[57]</td><td colspan="6">No</td><td colspan="3">Yes</td></tr>
<tr><td>Hardware-accelerated async-copy</td><td colspan="7" rowspan="4">No</td><td colspan="2" rowspan="4">Yes</td></tr>
<tr><td>Hardware-accelerated split arrive/wait barrier</td></tr>
<tr><td>Warp-level support for reduction ops</td></tr>
<tr><td>L2 cache residency management</td></tr>
<tr><td>DPX instructions for accelerated dynamic programming</td><td colspan="8" rowspan="4">No</td><td rowspan="4">Yes</td></tr>
<tr><td>Distributed shared memory</td></tr>
<tr><td>Thread block cluster</td></tr>
<tr><td>Tensor memory accelerator (TMA) unit</td></tr>
<tr><th rowspan="2">Feature support (unlisted features are supported for all compute capabilities)</th><th>1.0,1.1</th><th>1.2,1.3</th><th>2.x</th><th>3.0</th><th>3.2</th><th>3.5, 3.7, 5.x, 6.x, 7.0, 7.2</th><th>7.5</th><th>8.x</th><th>9.0</th></tr>
<tr><th colspan="9">Compute capability (version)</th></tr>
</table>

[Wikimedia: wikipedia.org/wiki/CUDA]

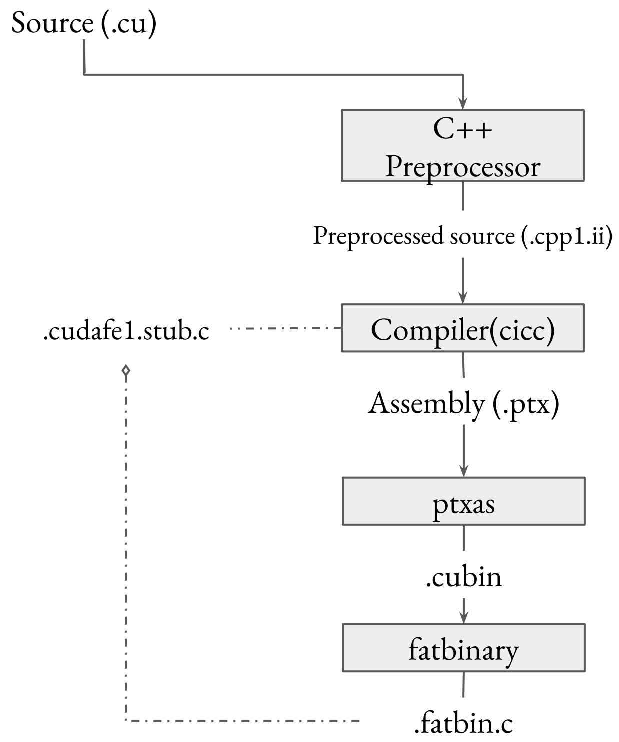
C build process



NVIDIA CUDA compilation process

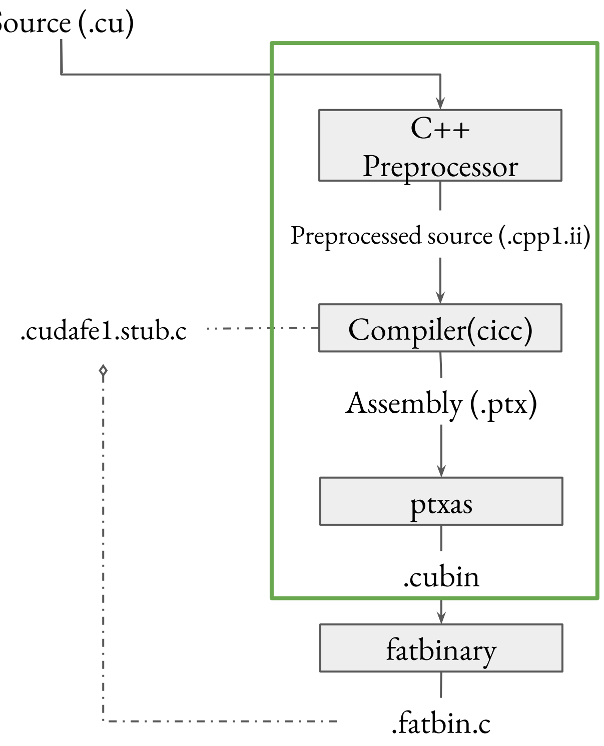
- NVIDIA CUDA compiler driver: `nvcc`
    - Performs the necessary steps to build a CUDA executable/library.
    - Includes compilation of both host code and device code.
- See https://docs.nvidia.com/cuda/cuda-compiler-driver-nvcc/index.html for documentation and more details.

CUDA build process



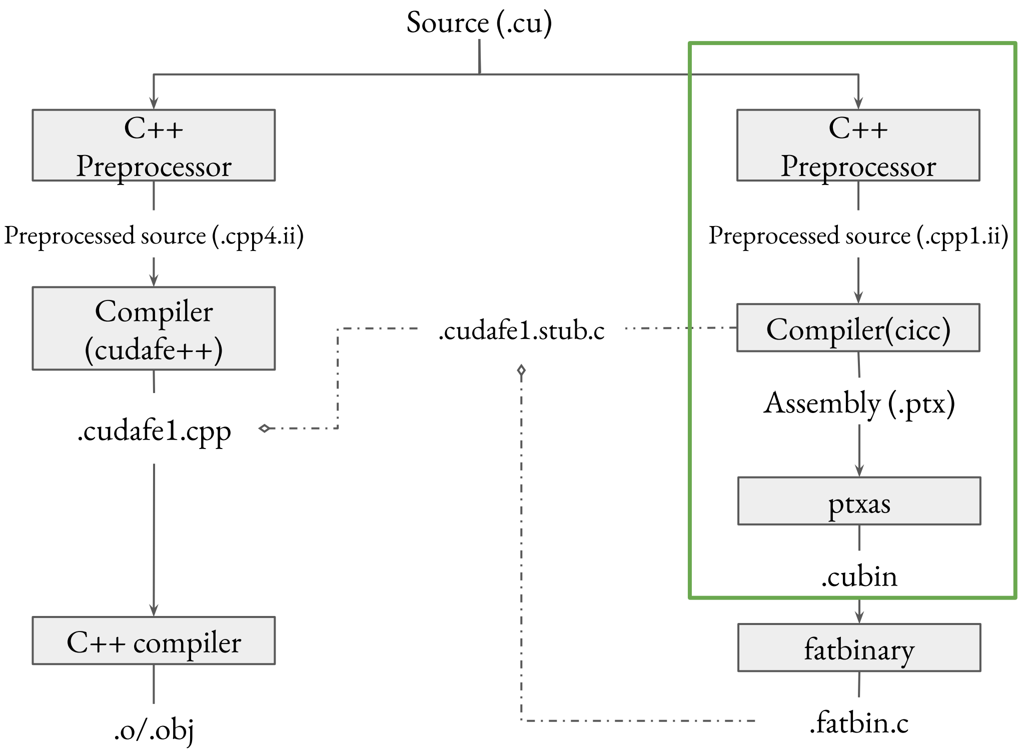
CUDA build process

- Repeat □ for each architecture



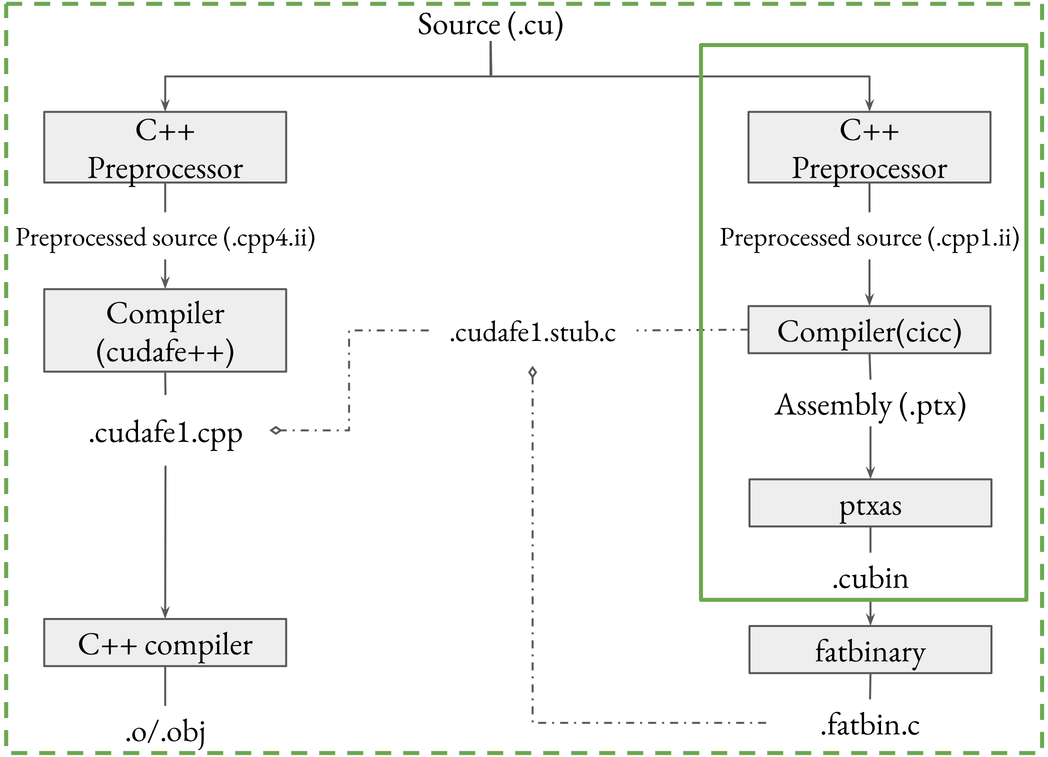
CUDA build process

- Repeat ☐ for each architecture



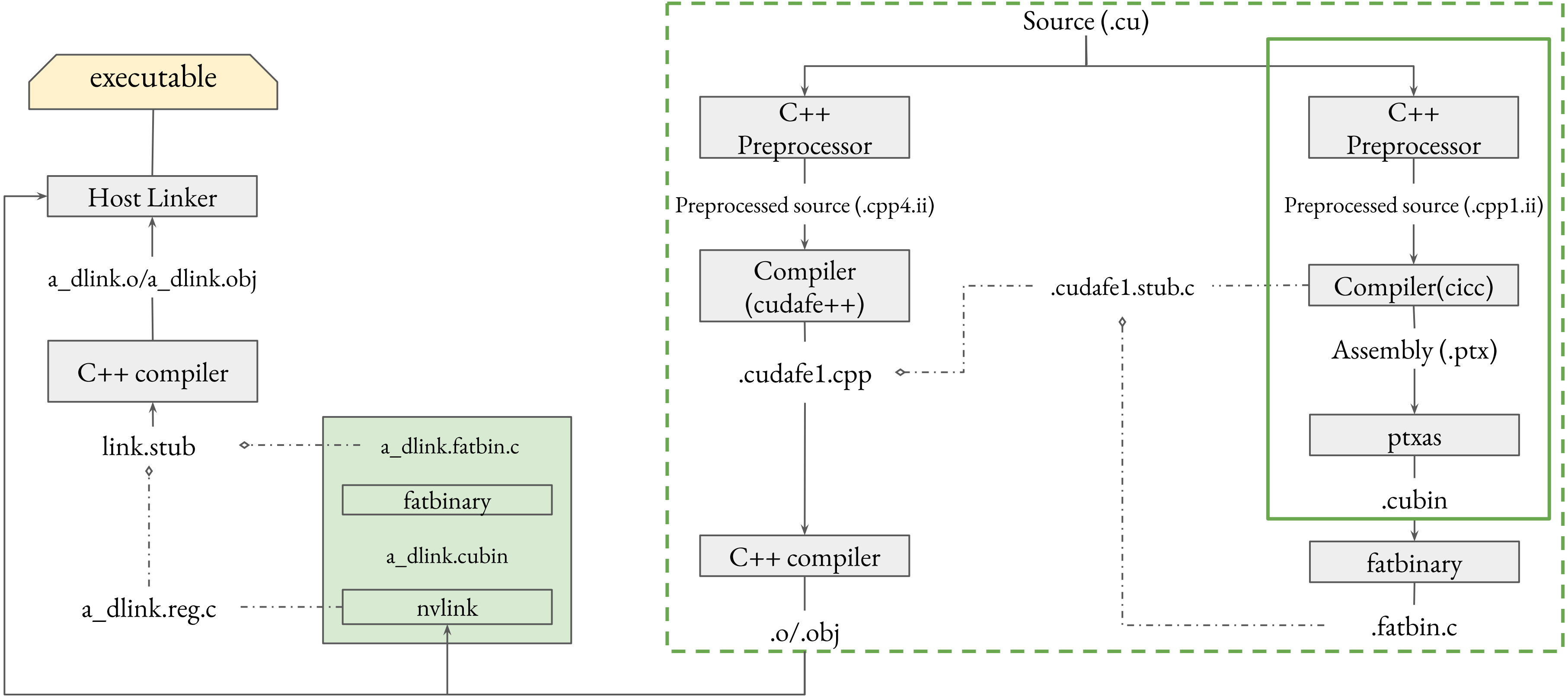
CUDA build process

- Repeat ☐ for each architecture
- Repeat ☐ for each .cu file



CUDA build process

- Repeat [solid green box] for each architecture
- Repeat [dashed green box] for each .cu file



Helpful tools (Linux)

- Dump assembly from object file:

```
objdump -d file.o                                # See documentation: man objdump
```

- Look at available symbols in the object file

```
nm file.o                                        # See documentation: man nm
```

- Look at linked libraries (dependencies on other shared objects)

```
ldd executable                                   # See documentation: man ldd
```


Measuring performance (best practices)

- Compilation flags are appropriate: For example with all the required optimization flags and no debug flags.
- Eliminate noise by repeating measurement a few times and averaging the overall time.
- Ensure operation is complete (synchronize explicitly if necessary).
- Ensure clock used is steady and cannot be arbitrarily updated (system_clock v/s steady_clock).
- Ensure operation is actually being performed (not skipped due to predication, or due to some clause).
- Try to eliminate common sources of interference: other users, some unrelated I/O, system effects.

Highly recommended read: https://blogs.fau.de/hager/archives/category/fooling-the-masses