# Data Flow Compilation for Greater Parallelism

- **Does not require new languages or ‘parallel programming’, only ‘restrict’ keyword for memory separation**
- **Replace traditional sequential compilation with DataFlow compilation**
- **Remove core specific L1 cache for ‘shared’ data**
- **Allow non-memory based core-to-core communication**

GUPS (random memory access) results

| | Intel Core i7-965 (3.2GHz) | Dual Intel Xeon E5620 (2.4GHz) | Dual AMD Opteron 6134 (2.3GHz) |
|---|---|---|---|
| gcc –m32 –O2 | 0.0359 GUPS | 0.0291 | 0.0127 |
| CHiMPS, 2 core | 0.0648 (1.8x) | 0.0493 (1.7x) | 0.0150 (1.2x) |
| CHiMPS, 4-core | 0.0886 (2.5x) | 0.0664 (2.3x) | 0.0206 (1.6x) |

DGEMM (sequential memory access) results

| | Intel Core i7-965 (3.2GHz) | Dual Intel Xeon E5620 (2.4GHz) | Dual AMD Opteron 6134 (2.3GHz) |
|---|---|---|---|
| gcc –m32 –O2 | 71 seconds | 85 seconds | 249 seconds |
| CHiMPS, 2 core | 62 seconds (1.14x) | 87 seconds (0.97x) | 181 seconds (1.3x) |
| CHiMPS, 4-core | 40 seconds (1.7x) | 68 seconds (1.25x) | 163 seconds (1.5x) |

Discrete (Finite Differential Solver) results

| | Intel Core i7-965 (3.2GHz) | Dual Intel Xeon E5620 (2.4GHz) | Dual AMD Opteron 6134 (2.3GHz) |
|---|---|---|---|
| gcc –m32 –O2 | 6.6 seconds | 6.6 seconds | 13 seconds |
| CHiMPS, 2 core | 2.9 seconds (2.27x) | 3.8 seconds (1.73x) | 7.7 seconds (1.68x) |
| CHiMPS, 4-core | 2.3 seconds (2.86x) | 2.7 seconds (2.44x) | 7.0 seconds (1.85x) |

- Results from compilation item only with traditional architectures
- Other items require new silicon

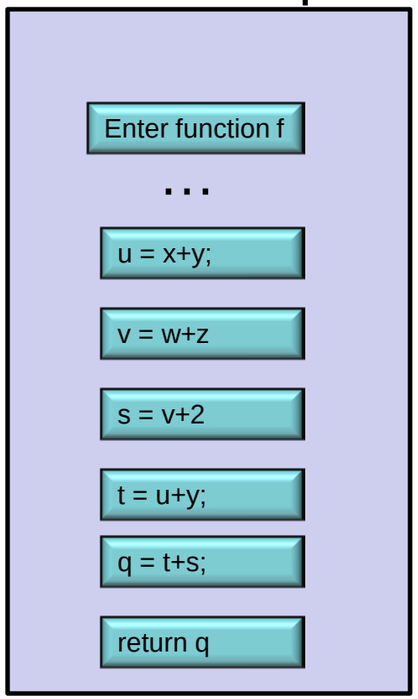
# Replace traditional sequential compilation with DataFlow compilation

## Source code

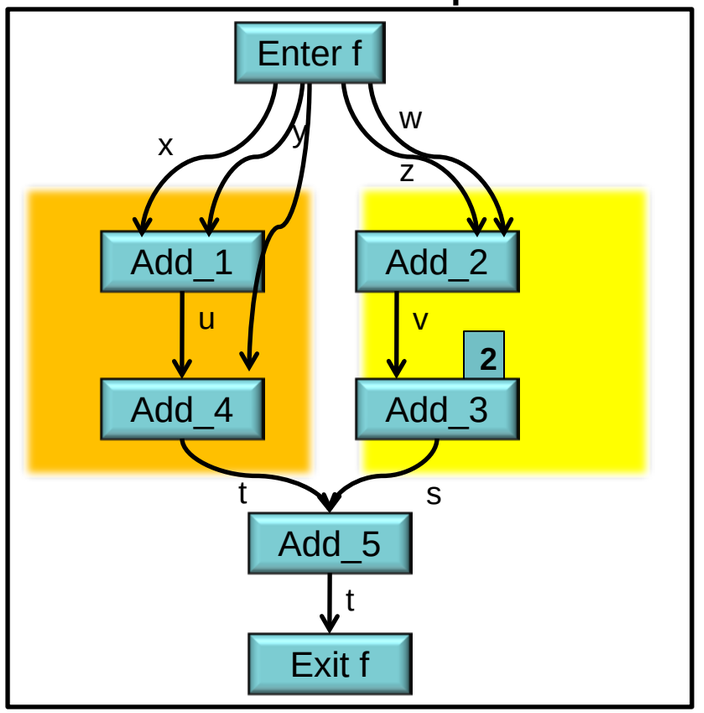
```
long f(long w, long x,
       long y, long z)
{
  long u = x+y;
  long v = w+z;
  long s = v+2;
  long t = u+y;
  long q = t+s;

  return q;
}
```

## Traditional compilation



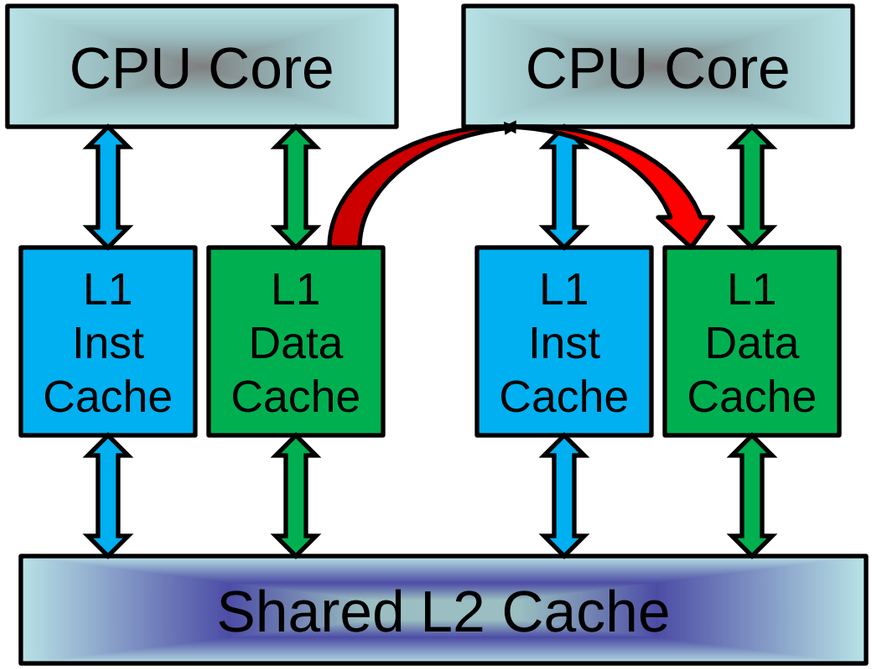

## DataFlow compilation



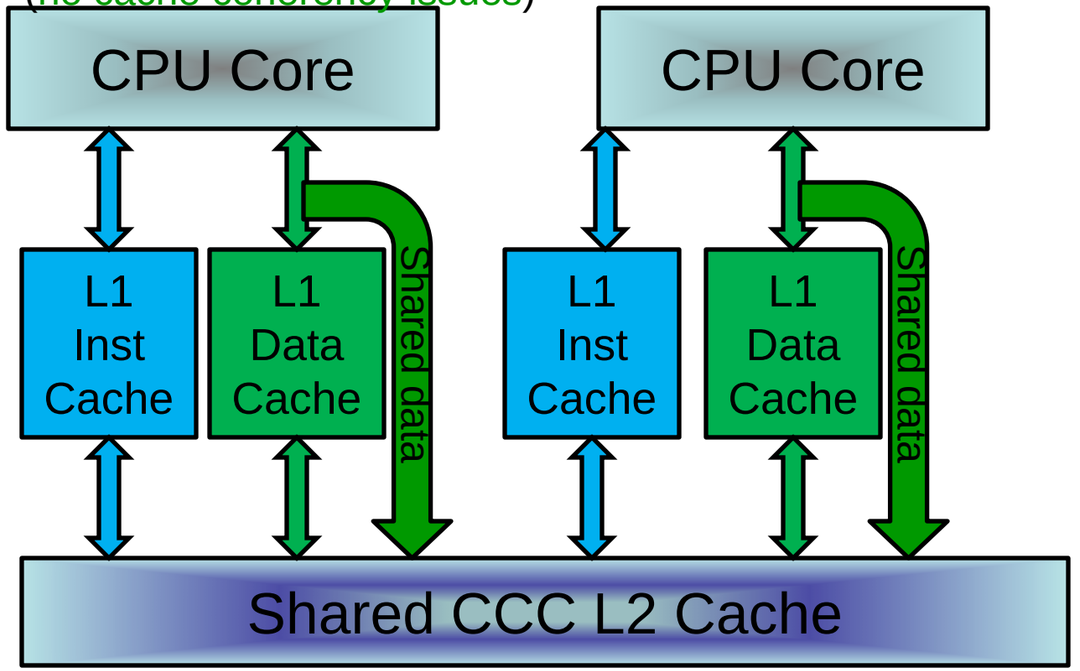

- **Allow each instruction or group of instructions to execute on a separate core (yellow and orange)**
- **Requires core-to-core communication for partial results**
- **Dataflow graph allows optimizations that are not apparent in traditional control flow graph (e.g. memory parallelism)**
- **Far more parallelism can be extracted than possible by human intervention**

# Remove core specific L1 cache for ‘shared’ data

Traditional architecture
(cache coherency issues for shared data)



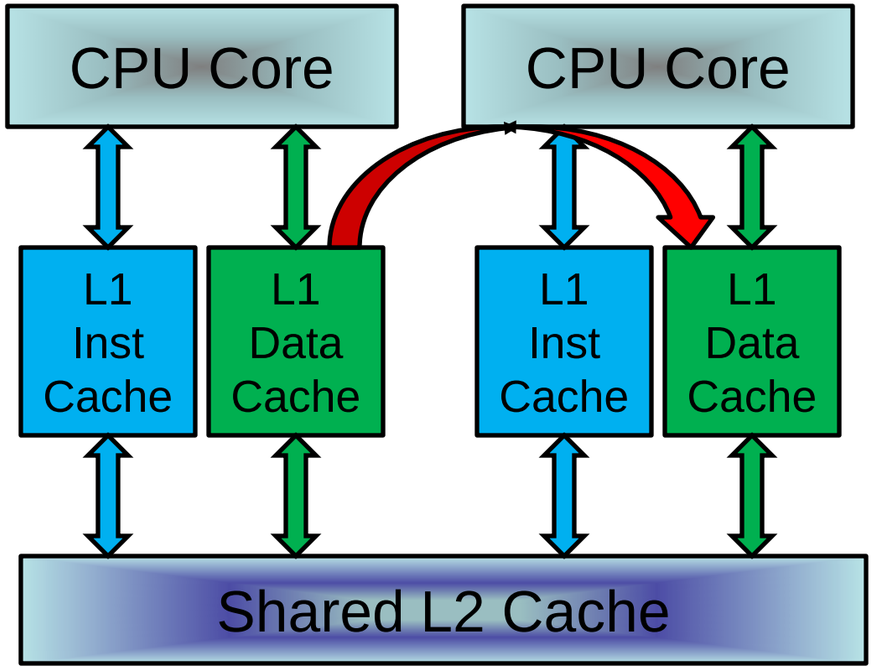

Modified architecture: L1 cache not used for shared data
(no cache coherency issues)



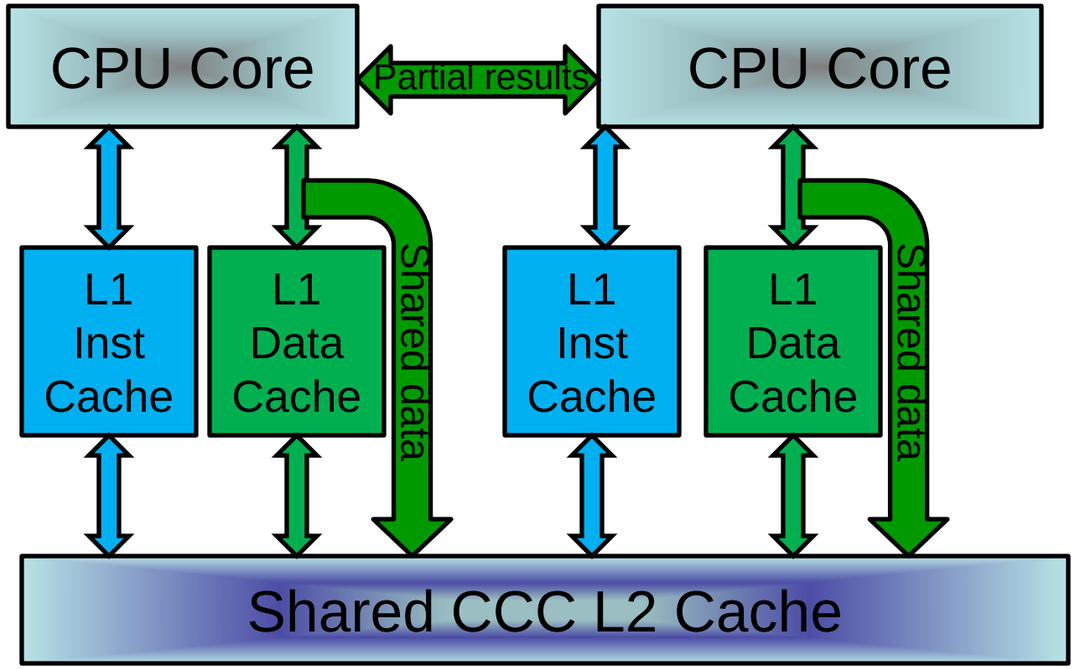

- **Shared data : at least one core has write access and another core has read and/or write access**
- **L2 CCC cache:  Crossbar connected cache with banking for better multicore access**

# Allow non-memory based core-to-core communication

Traditional architecture
(core-to-core data transfer goes through L1/L2 cache as shared data)




Direct core-to-core communication
(used for partial results which need never live in memory)




- **Partial results computed in one core, passed to another core directly.**
- **Does not affect (fill) caches.**
- **Temporary data that never lives in ‘off-chip’ memory.**