องค์ประกอบของเครื่องคอมพิวเตอร์ และภาษาแอสเซมบลี: ARM และ RaspberryPi3

บทที่ 8 การคำนวณแบบขนานด้วยบอร์ด Pi

รศ.ดร.สุรินทร์ กิตติธรกุล
ภาควิชาวิศวกรรมคอมพิวเตอร์ คณะวิศวกรรมศาสตร์
สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

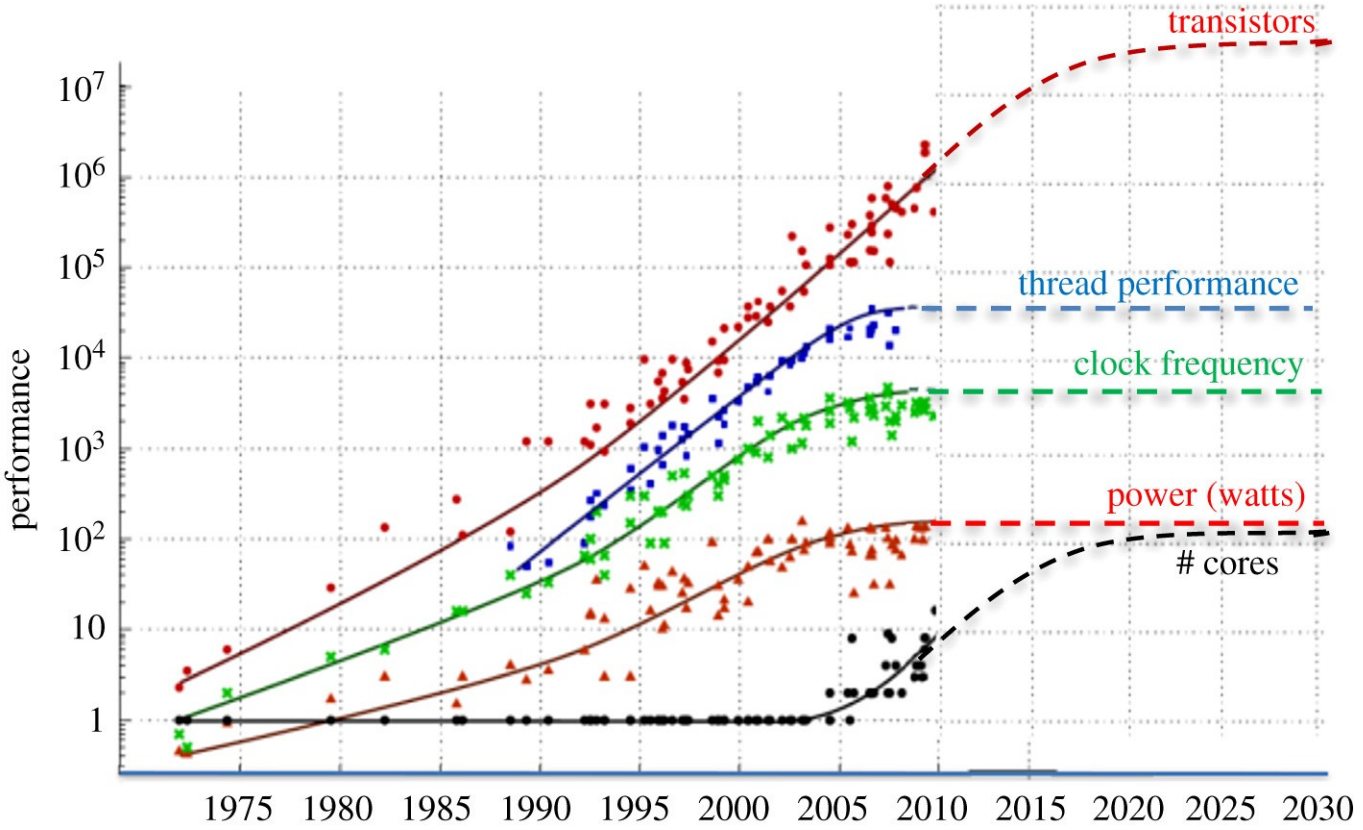
สารบัญ

- บทที่ 1 บทนำ
- บทที่ 2 ข้อมูลและคณิตศาสตร์ในคอมพิวเตอร์
- บทที่ 3 ฮาร์ดแวร์และซอฟต์แวร์ของคอมพิวเตอร์
- บทที่ 4 ภาษาแอสเซมบลีของ ARM เวอร์ชัน 32 บิท
- บทที่ 5 ลำดับชั้นของหน่วยความจำ
- บทที่ 6 อุปกรณ์/วงจรอินพุตและเอาท์พุต
- บทที่ 7 อุปกรณ์เก็บรักษาข้อมูลและระบบไฟล์
- บทที่ 8 การคำนวณแบบขนานด้วยบอร์ด Pi3

8 การคำนวณแบบขนาน

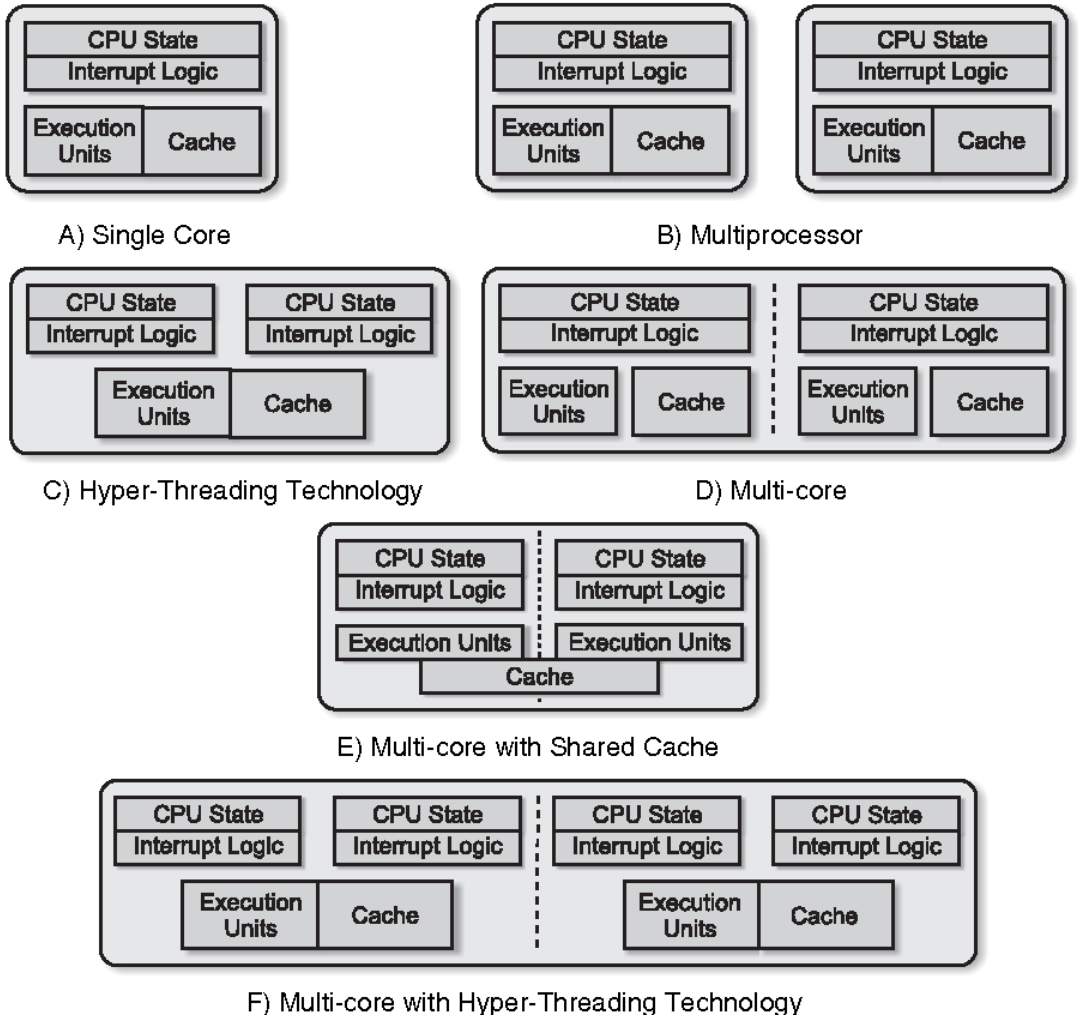
จำนวนทรานซิสเตอร์และประสิทธิภาพของซีพียูเพิ่มขึ้นเฉลี่ยไม่มาก ประมาณ 10 เท่าในเวลา 10 ปี จากเส้นกราฟในรูปที่ M.1 แต่ในช่วงปี ค.ศ. 1995-2005 จำนวนทรานซิสเตอร์และประสิทธิภาพเพิ่มขึ้นเฉลี่ยเป็น 2 เท่าทุก 1.5 ปีหรือ 10 เท่าใน 5 ปีโดยประมาณตามกฎของ Moore (Moore's Law) ที่มา: wikipedia สืบเนื่องจากการเพิ่มจำนวนบิตข้อมูลที่ประมวลผลได้จาก 4 บิตเป็น 8, 16, 32 และ 64 บิตต่อ 1 คาบสัญญาณคล็อก จากการปรับโครงสร้างการทำงานจากไปป์ไลน์เป็นซูเปอร์สเกลาร์ (SuperScalar) ซึ่งเป็นการเริ่มต้นของการทำงานแบบขนานระดับคำสั่ง (Instruction Level Parallelism) เป็นการทำงานแบบขนานระดับเธรด (Thread Level Parallelism) ในชิปที่รองรับมัลติเธรด (Multithread) และกลายเป็นชนิดมัลติคอร์ (Multi-Core) ที่มา: wikipedia จนถึงชนิดแมนีคอร์ (Many Core) ที่มา: wikipedia ในปัจจุบันและอนาคตอันใกล้

8 การคำนวณแบบขนาน



8 การคำนวณแบบขนาน

- จำนวนทรานซิสเตอร์ต่อชิปมีจำนวนเพิ่มขึ้นจากชิปตัวแรกจนถึงจุดอิ่มตัวที่จำนวน 10^8 เท่า หรือประมาณหลายพันล้านตัวในปัจจุบัน และมีแนวโน้มเพิ่มขึ้นตามจำนวนคอร์ที่เพิ่มขึ้น
- ความถี่สัญญาณคล็อกสูงสุด หน่วยเป็น เมกะเฮิรตซ์ เพิ่มขึ้นช้าประมาณ 10 เท่าในเวลา 12-15 ปีและเพิ่มขึ้นแบบก้าวกระโดดประมาณ 10 เท่าในเวลา 5-7 ปีและถึงจุดอิ่มตัวในปี ค.ศ. 2005 เป็นต้นมา
- ค่าประสิทธิภาพของซีพียูเพียง 1 เธรด คือ ประสิทธิภาพของเธรดเดี่ยว (Single Thread Performance) เพิ่มขึ้นอย่างก้าวกระโดดตามความถี่สัญญาณคล็อกและจำนวนทรานซิสเตอร์ เนื่องจากสามารถใช้งานซีพียูได้เพียง 1 แกนประมวลผลเท่านั้น และแนวโน้มจะเปลี่ยนแปลงน้อย ประสิทธิภาพโดยรวมของซีพียูจะเพิ่มขึ้นเนื่องจากจำนวนคอร์
- จำนวนแกนประมวลผลมีเพียง 1 แกนประมวลผลต่อชิปจากอดีตและเพิ่มขึ้นอย่างอย่างก้าวกระโดดและจะเพิ่มขึ้นอย่างต่อเนื่องจนอาจจะอิ่มตัวที่จำนวน 128 แกนประมวลผลโดยประมาณ
- ค่ากำลังไฟสูงสุด (Power) หน่วยเป็นวัตต์ มีค่าเพิ่มขึ้นตามความถี่สัญญาณคล็อกสูงสุด จะมีแนวโน้มค่อนข้างคงที่เพราะข้อจำกัดด้านการระบายความร้อนออกจากตัวชิป แนวโน้มการบริโภคพลังงานที่ไม่เพิ่มสูงอาจเนื่องมาจากเทคโนโลยีการผลิตที่พัฒนาในอนาคต

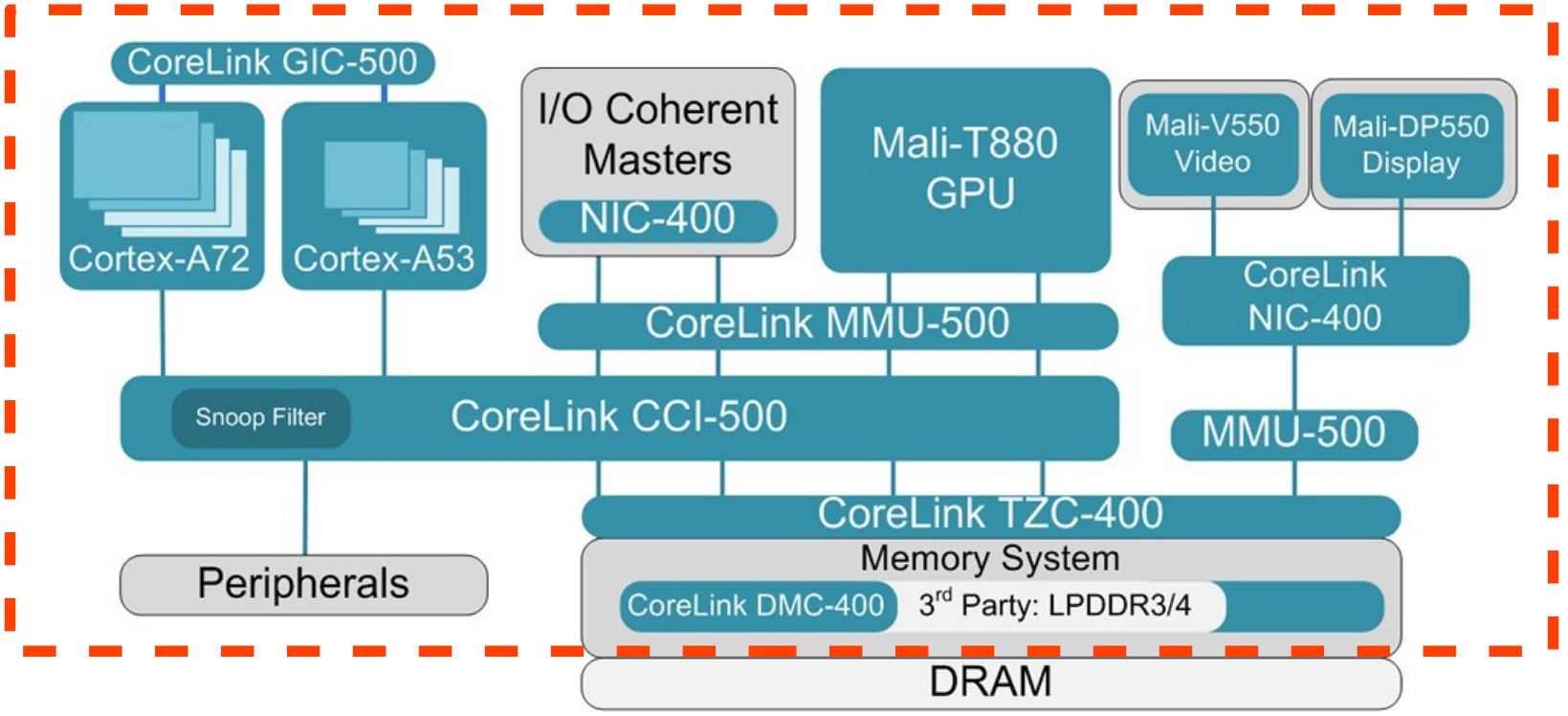


8.1 เครื่องคอมพิวเตอร์แบบขนานชนิดมัลติคอร์ต่อชิป

8.1 เครื่องคอมพิวเตอร์แบบขนานชนิดมัลติคอร์ต่อชิป

- เครื่องคอมพิวเตอร์แบบขนานชนิด **SMP** (Shared Memory Multiprocessor) ทุกๆ แกนประมวลผลใช้หน่วยความจำหลักหรือ DRAM และอินพุตเอาต์พุตร่วมกัน เป็นต้นแบบของซีพียูมัลติคอร์ในเครื่องคอมพิวเตอร์ทุกชนิดปัจจุบัน ซีพียูเหล่านี้มาจากของผู้ผลิตรายสำคัญ เช่น Intel, AMD และ ARM เป็นหลัก กรณีศึกษาในหัวข้อที่ 8.1.1 และ 8.1.2
- **มัลติคอมพิวเตอร์** (Multicomputer) Patterson and Hennessy (2016) มีลักษณะสำคัญคือ การนำคอมพิวเตอร์ชนิด SMP มาช่วยกันแก้ปัญหาเดียวกัน แต่ละเครื่องมีระบบปฏิบัติการที่แตกต่างกันหรือเหมือนกันก้อได้ ทำให้แต่ละใช้หน่วยความจำ และ I/O แยกอิสระจากกัน การทำงานของโปรแกรมบนเครื่องคอมพิวเตอร์ใช้การส่งผ่านข้อความระหว่างกัน (Message Passing) ผ่านเครือข่ายเพื่อประสานงานและสื่อสารกัน มัลติคอมพิวเตอร์สามารถจำแนกออกเป็นชนิดย่อยๆ คือ

8.1.1 กรณีศึกษา ARM Cortex A72 และ Cortex A53



8.1.1 กรณีศึกษา ARM Cortex A72 และ Cortex A53

โครงสร้างของชิปซีพียูสำหรับคอมพิวเตอร์เคลื่อนที่ชนิดแท็บเล็ต สมาร์ทโฟน และระบบสมองกลฝังตัวที่ได้รับความนิยม อุปกรณ์เหล่านี้ประกอบด้วยซีพียู ARM Cortex A72 จำนวน 4 แกนประมวลผลและ Cortex A53 จำนวน 4 แกนประมวลผลดังรูปที่ 8.2 เพื่อทำงานร่วมกันตามหลักการ **big.LITTLE** ของบริษัท ARM โดยซีพียู big คือ Cortex A72 เนื่องจากประสิทธิภาพสูงทำให้บริโภคพลังงานมากกว่าเหมาะสำหรับเกมและมัลติมีเดียคุณภาพสูง และซีพียู LITTLE คือ Cortex A53 เพราะมีประสิทธิภาพต่ำทำให้บริโภคพลังงานน้อยกว่าเหมาะสำหรับโปรแกรมทั่วไปที่ใช้งานปกติ ซีพียูทั้งหมดเชื่อมต่อกันด้วย Cache Coherence Interconnect ตัวอย่างชิปที่ใช้ซีพียูทั้งสองชนิดนี้ ได้แก่ ชิป Helio X20 ที่มา: wikichip.org ผลิตโดยบริษัท MediaTek ชิป RK3399 ที่มา: wikidot.com ผลิตโดยบริษัท Rockchip สำหรับคอมพิวเตอร์บอร์ดเดี่ยว Rock Pi 4 ที่มา: rockpi.org และ ชิป i.MX8 NXP ที่มา: nxp.com ผลิตโดยบริษัท NXP สำหรับบอร์ดพัฒนาระบบฝังตัว ที่มา: variscite.com เป็นต้น

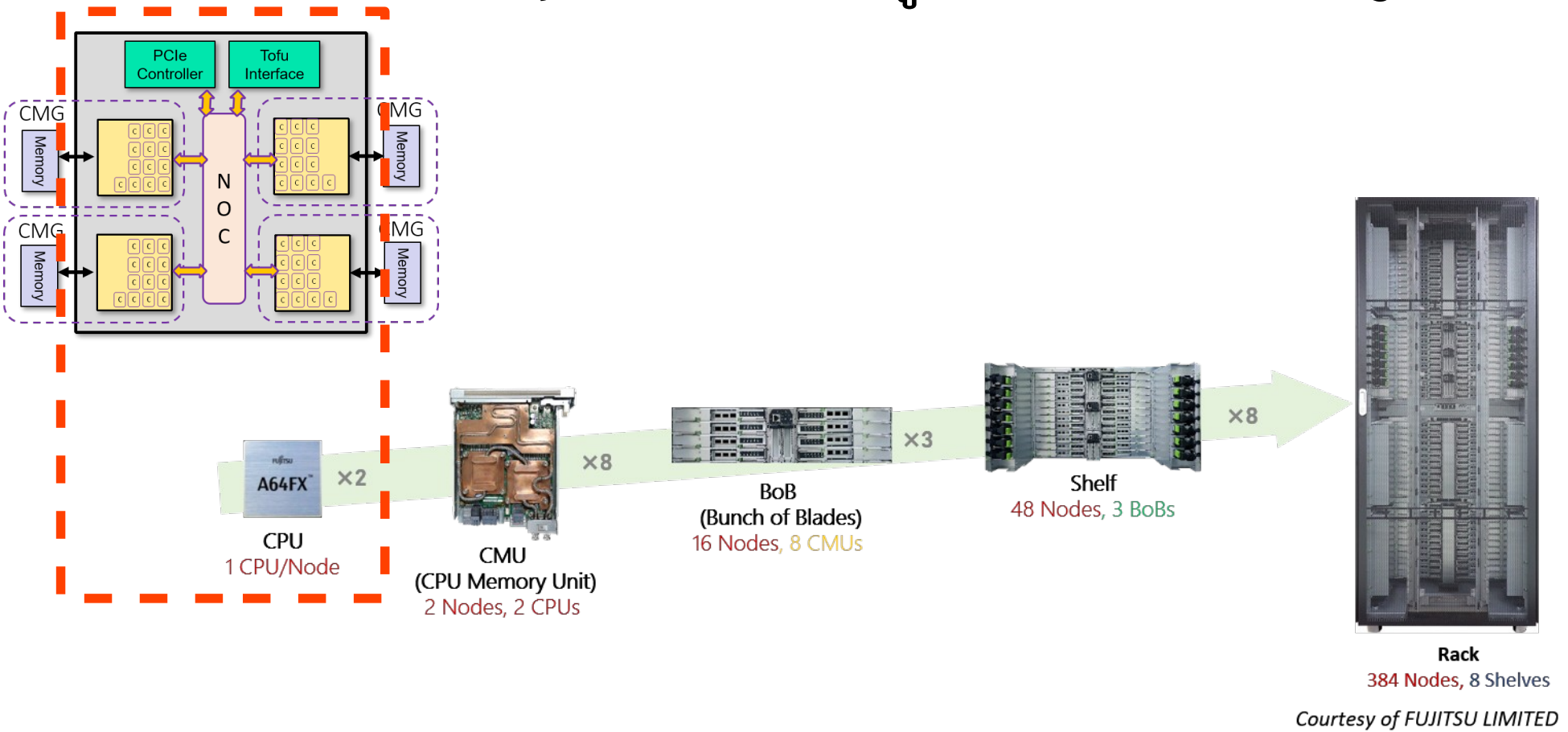
SUPERCOMPUTER FUGAKU - SUPERCOMPUTER FUGAKU, A64FX 48C 2.2GHZ, TOFU INTERCONNECT D

Site:	RIKEN Center for Computational Science
System URL:	https://www.r-ccs.riken.jp/en/fugaku/project
Manufacturer:	Fujitsu
Cores:	7,630,848
Memory:	5,087,232 GB
Processor:	A64FX 48C 2.2GHz
Interconnect:	Tofu interconnect D
Performance	
Linpack Performance (Rmax)	442,010 TFlop/s
Theoretical Peak (Rpeak)	537,212 TFlop/s
Nmax	21,288,960
HPCG [TFlop/s]	16,004.5
Power Consumption	
Power:	29,899.23 kW (Optimized: **26248.36** kW)

8.1.2 กรณีศึกษาชิป Fujitsu A64FX ในซูเปอร์คอมพิวเตอร์ Fugaku

https://www.top500.org/system/179807/

8.1.2 กรณีศึกษา Fujitsu A64FX ในซูเปอร์คอมพิวเตอร์ Fugaku

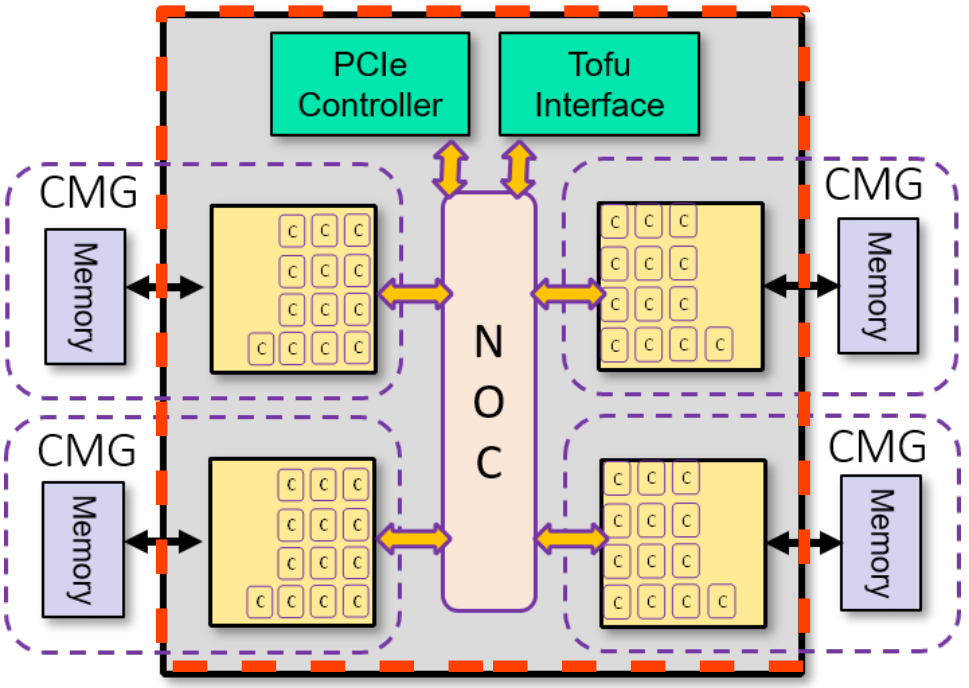


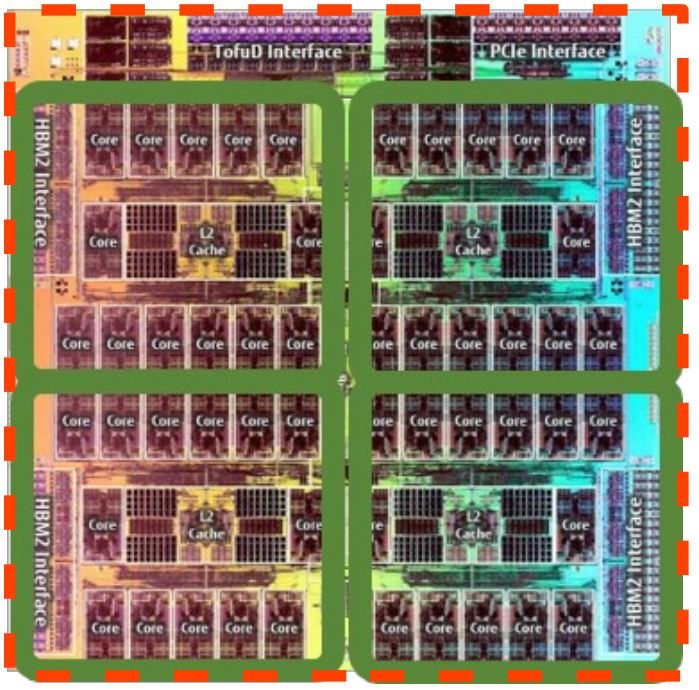
8.1.2 กรณีศึกษา Fujitsu A64FX ในซูเปอร์คอมพิวเตอร์ Fugaku

รูปที่ 8.4 แสดงให้เห็นโครงสร้างภายในของซีพียู Fujitsu A64FX ประกอบด้วยซีพียู ARM จำนวน 4 ชุดๆ ละ 12 แกนประมวลผลรวม 48 แกนประมวลผล ทำงานที่ความถี่ 2.2 GHz ซีพียูแต่ละแกนประมวลผลรองรับคำสั่ง ARM เวอร์ชัน v8.2-A 64 บิต SVE (Scalable Vector Extension) โดย Fujitsu และ ARM ออกแบบร่วมกัน ผู้อ่านสามารถค้นคว้ารายละเอียดเพิ่มเติมที่ wikipedia

ซีพียูแต่ละชุดเชื่อมต่อกันเครือข่ายบนชิป (Network on Chip: NoC) ชนิดบัสวงแหวน (Ring Bus) และเชื่อมต่อกับหน่วยความจำ HBM2 (High Bandwidth Memory 2) รวม 32 กิกะไบต์ รายละเอียดเพิ่มเติมที่ wikipedia ตัวชิปเชื่อมต่อกันระหว่างชิปด้วยวงจรชื่อ ToFuD Interface ผ่านอินเตอร์เฟส PCIe (Peripheral Component Interconnect Express) ชิป Fujitsu A64FX ผลิตโดยบริษัท TSMC (Taiwan Semiconductor Manufacturing Company ด้วยเทคโนโลยี 7 นาโนเมตร ผู้อ่านสามารถค้นคว้ารายละเอียดเกี่ยวกับชิปเพิ่มเติมที่ github.com

8.1.2 กรณีศึกษา Fujitsu A64FX ในซูเปอร์คอมพิวเตอร์ Fugaku



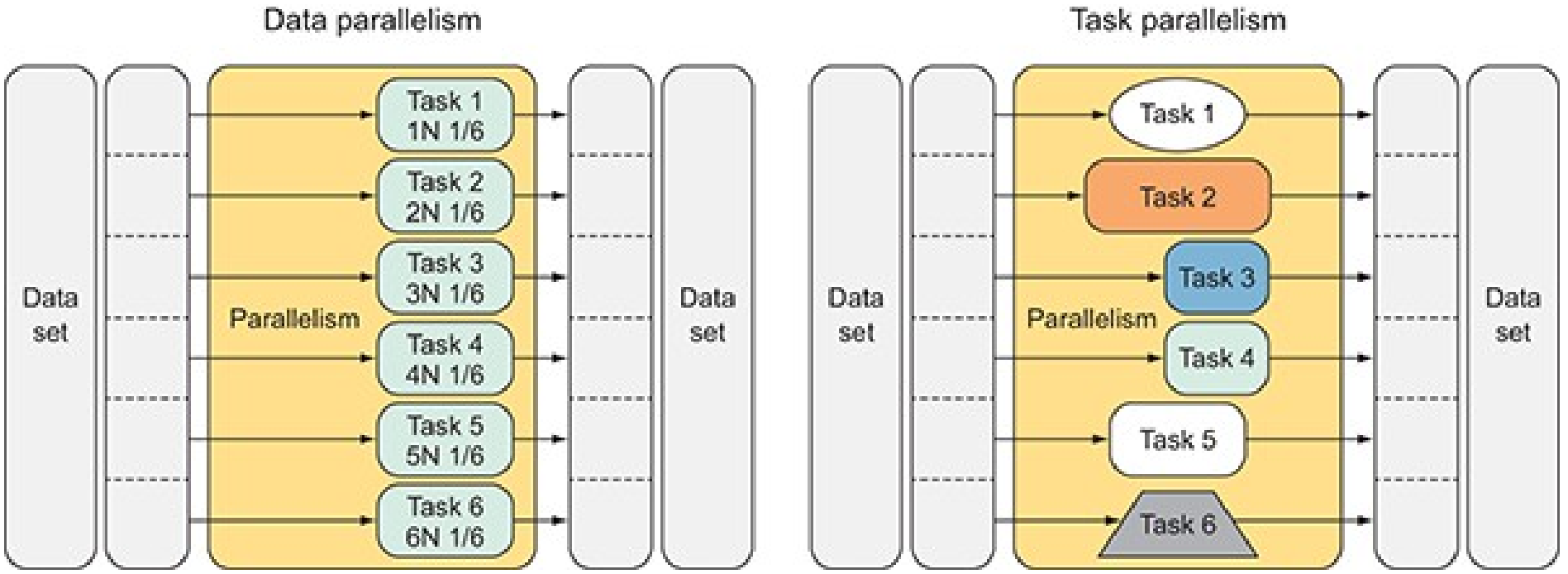


8.2 ความขนาน (Parallelism)

การคำนวณแบบขนานตั้งอยู่บนพื้นฐานของฮาร์ดแวร์และความขนาน (Parallelism) ของปัญหา การคำนวณแบบขนาน ประกอบด้วยองค์ประกอบสำคัญ 2 ส่วนคือ

- อัลกอริทึมแบบขนาน (Parallel Algorithm) สำหรับแก้ปัญหาโจทย์ต่างๆ ที่สามารถใช้ประโยชน์จากความขนานของปัญหาหรือโจทย์นั้นๆ
- เครื่องคอมพิวเตอร์แบบขนาน (Parallel Computer) ชนิดมัลติคอร์และชนิดแมนีคอร์ซึ่งใช้หน่วยความจำกายภาพร่วมกันมีประสิทธิภาพสูงขึ้นและราคาต่ำลง การพัฒนาโปรแกรมแบบขนานบนเครื่องคอมพิวเตอร์ชนิดนี้อาศัยการพัฒนาโปรแกรมแบบมัลติเธรด (Multithread Programming) ด้วยไลบรารีมาตรฐาน เช่น Pthread และ OpenMP

8.2 ความขนาน (Parallelism)



8.3 การพัฒนาอัลกอริทึมแบบมัลติเธรดและแบบขนานด้วยไลบรารี OpenMP

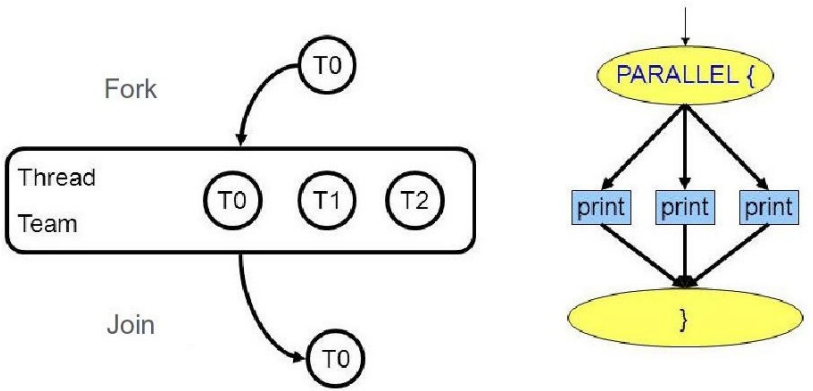
ในปัจจุบัน การพัฒนาอัลกอริทึมแบบขนานมีความสำคัญมากขึ้น เนื่องจากอุปกรณ์คอมพิวเตอร์มีซีพียูชนิดมัลติคอร์และชนิดแมนีคอร์เพิ่มมากขึ้นเรื่อยๆ ตามกรณีศึกษาที่กล่าวไปแล้ว ดังนั้น โปรแกรมเมอร์สามารถแปลงฟังก์ชันที่ถูกเรียกใช้บ่อยๆ ให้กลายเป็นเธรดของการทำงานบนคอร์พร้อมๆ กันได้ โดยมีระบบปฏิบัติการที่รองรับการทำงานแบบ**มัลติเธรด** ภาษาที่รองรับการทำงานแบบมัลติเธรด ได้แก่ ภาษา C/C++ ใช้ไลบลารี (Library) มาช่วยเสริมจึงจะรองรับการทำงานแบบมัลติเธรด เช่น PThread และ OpenMP เป็นต้น ส่วนภาษา JAVA รองรับการทำมัลติเธรดโดยธรรมชาติ โดยสรุป โปรแกรมต่างๆ ที่มีอัลกอริทึมแบบขนานสามารถทำงานอยู่บนซีพียูชนิดมัลติคอร์ จะต้องสร้างจำนวนเธรดให้มากกว่าหรือเท่ากับจำนวนคอร์ที่มี เพื่อกระจายเธรดต่างๆ ไปยังทุกๆ แกนประมวลผลให้ทำงานพร้อมๆ กัน

8.3 การพัฒนาอัลกอริทึมแบบมัลติเธรดและแบบขนานด้วยไลบรารี OpenMP

ไลบรารี **openMP** มีประโยคโครงสร้าง (Construct) ที่เสริมการพัฒนาโปรแกรมคอมพิวเตอร์ภาษา C/C++ และภาษา Fortran ให้สามารถทำงานแบบมัลติเธรดได้ โดยองค์กร openmp.org ได้กำหนดเป็นมาตรฐาน แต่ละเวอร์ชันให้คอมไพเลอร์แต่ละตัวทำหน้าที่แปลซอร์สโค้ดตามมาตรฐานด้วยความสมัครใจ ซึ่งตำราเล่มนี้ใช้คอมไพเลอร์ GCC เป็นเครื่องมือหลัก

```
#pragma omp parallel ...
{
// Parallel Region
}
```

```
#pragma omp parallel
{
 printf("Hello world %d\n", omp_get_thread_num());
}
```



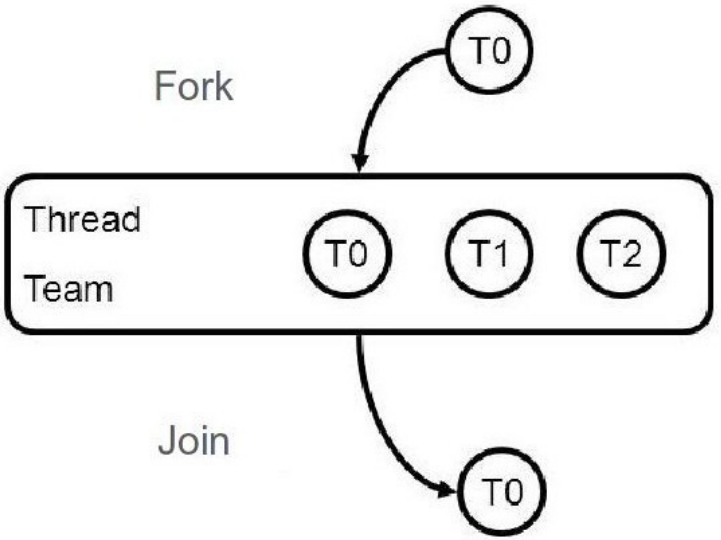
8.3 การพัฒนาอัลกอริทึมแบบมัลติเธรดและแบบขนานด้วยไลบรารี OpenMP

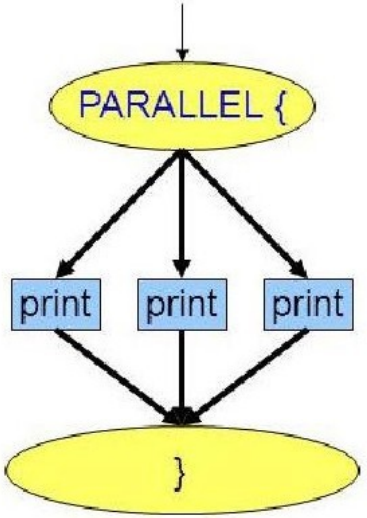
ทุกโปรแกรมเริ่มต้นรันด้วยการสร้าง**เธรดหลัก** หรือ (**Master Thread**) เพื่อทำงานแบบอนุกรม (Serial) ก่อนเสมอ เมื่อเธรดหลักรันมาถึงโปรแกรมบริเวณ Parallel Region เธรดหลักสามารถสร้างเธรดผู้ช่วย ตามรายละเอียดของประโยคนั้น และแจกจ่ายภาระงานให้เธรดเสริมและตัวเธรดหลักเองช่วยกันทำงาน ขั้นตอนนี้เรียกว่า การ **Fork** ในรูปที่ 8.6 เมื่อแต่ละเธรดทำงานเสร็จสิ้นแล้วก็จะรวมกับเธรดหลัก เรียกว่า การ **Join** เหลือเพียงเธรดหลักทำงานต่อแบบอนุกรมจนกว่าเธรดหลักจะรันถึงบริเวณ Parallel Region อื่นๆ อีก ไลบรารี OpenMP เรียกการทำงานลักษณะนี้ว่า โมเดล Fork-Join

ตามซอร์สโค้ดตัวอย่างในรูปที่ 8.6 การทำงานแบบขนานของเธรดรวม 3 เธรด ประกอบด้วยเธรด T0, T1 และ T2 สั่งพิมพ์ข้อความ Hello World ตามด้วยหมายเลขเธรดประจำตัว โดยใช้ไลบรารี OpenMP เธรด **T0** หมายถึง **เธรดหลัก**ซึ่งมีหมายเลขเธรดประจำตัว (Thread ID) **เท่ากับ 0 เสมอ** ส่วน T1 และ T2 คือ **เธรดผู้ช่วย** (Worker Thread) ซึ่งโปรแกรมเมอร์สามารถอ่านค่าหมายเลขเธรดประจำตัวของแต่ละเธรดได้

8.3 การพัฒนาอัลกอริทึมแบบมัลติเธรดและแบบขนานด้วยไลบรารี OpenMP

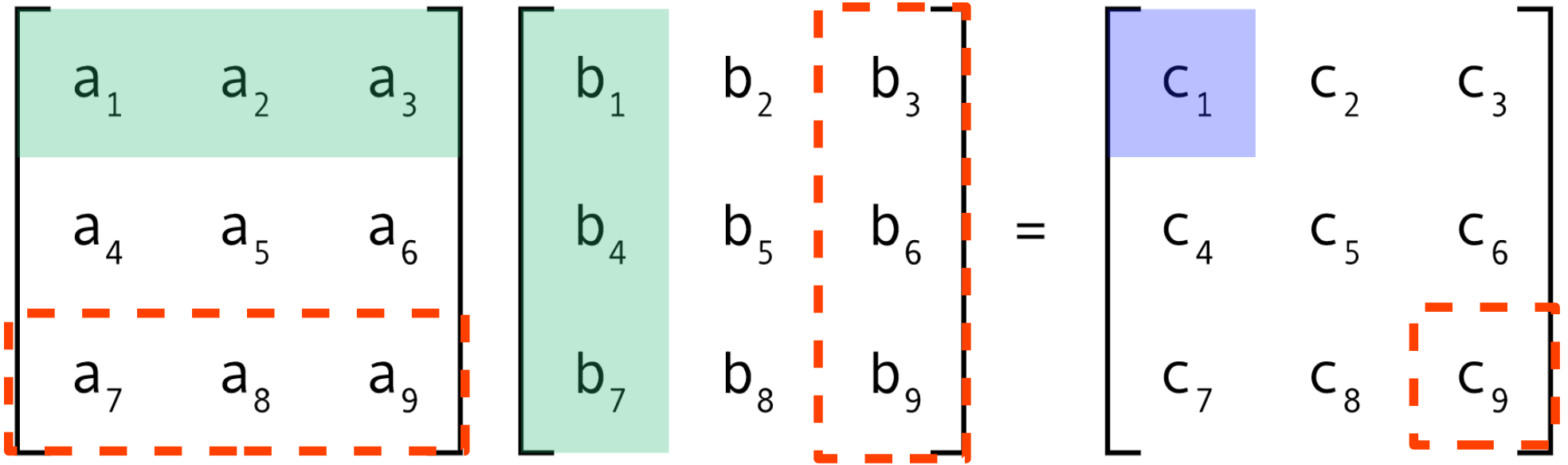
```
#pragma omp parallel
{
 printf("Hello world %d\n", omp_get_thread_num());
}
```






```
Hello world 1
Hello world 0
Hello world 2
```

8.3.1 การคูณแมทริกซ์ (Matrix Multiplication) แบบขนาน



8.3.1 การคูณแมทริกซ์ (Matrix Multiplication) แบบขนาน

```
#pragma omp parallel for private(k, j)
  for(i=0;i<N;i++) {
    for(j=0;j<N;j++) {
      C[i][j]=0.; // set initial value of resulting matrix C = 0
      for(k=0;k<N;k++) {
        C[i][j]=C[i][j]+A[i][k]*B[k][j];
      } // k
    } // j
  } // i
```

หมายเหตุ i คือหมายเลขแถวของแมทริกซ์ C และ j คือหมายเลขคอลัมน์ของแมทริกซ์ C

8.3.1 การคูณแมทริกซ์ (Matrix Multiplication) แบบขนาน

การทดลองจับเวลาเฉพาะการวนรอบนี้เพื่อเปรียบเทียบและคำนวณต่างๆ เธรดหลักจะสร้างเธรดผู้ช่วยที่ประโยคโครงสร้าง **#pragma omp parallel for** ตามจำนวนเธรดที่ผู้ใช้ต้องการ ยกตัวอย่างเช่น 4 เธรด นับรวมเธรดหลัก ตามหลักการ SMP แต่ละเธรดซึ่งประจำอยู่ที่ซีพียูแต่ละแกนประมวลผลจะมองเห็นค่าของตัวแปรทุกตัว ยกเว้น j และ k ซึ่งจะอธิบายต่อไป

เธรดหลักจะแบ่งข้อมูลเท่ากันให้เธรดทั้ง 4 เธรดตามตัวนับรอบ i ดังนี้

- i=0 ถึง $\frac{N}{4}$-1 เพื่อคำนวณ $C[i][j]$ โดย j=0 ถึง N-1
- i=$\frac{N}{4}$ ถึง $\frac{N}{2}$-1 เพื่อคำนวณ $C[i][j]$ โดย j=0 ถึง N-1
- i=$\frac{N}{2}$ ถึง $\frac{3N}{4}$-1 เพื่อคำนวณ $C[i][j]$ โดย j=0 ถึง N-1
- i=$\frac{3N}{4}$ ถึง N-1 เพื่อคำนวณ $C[i][j]$ โดย j=0 ถึง N-1

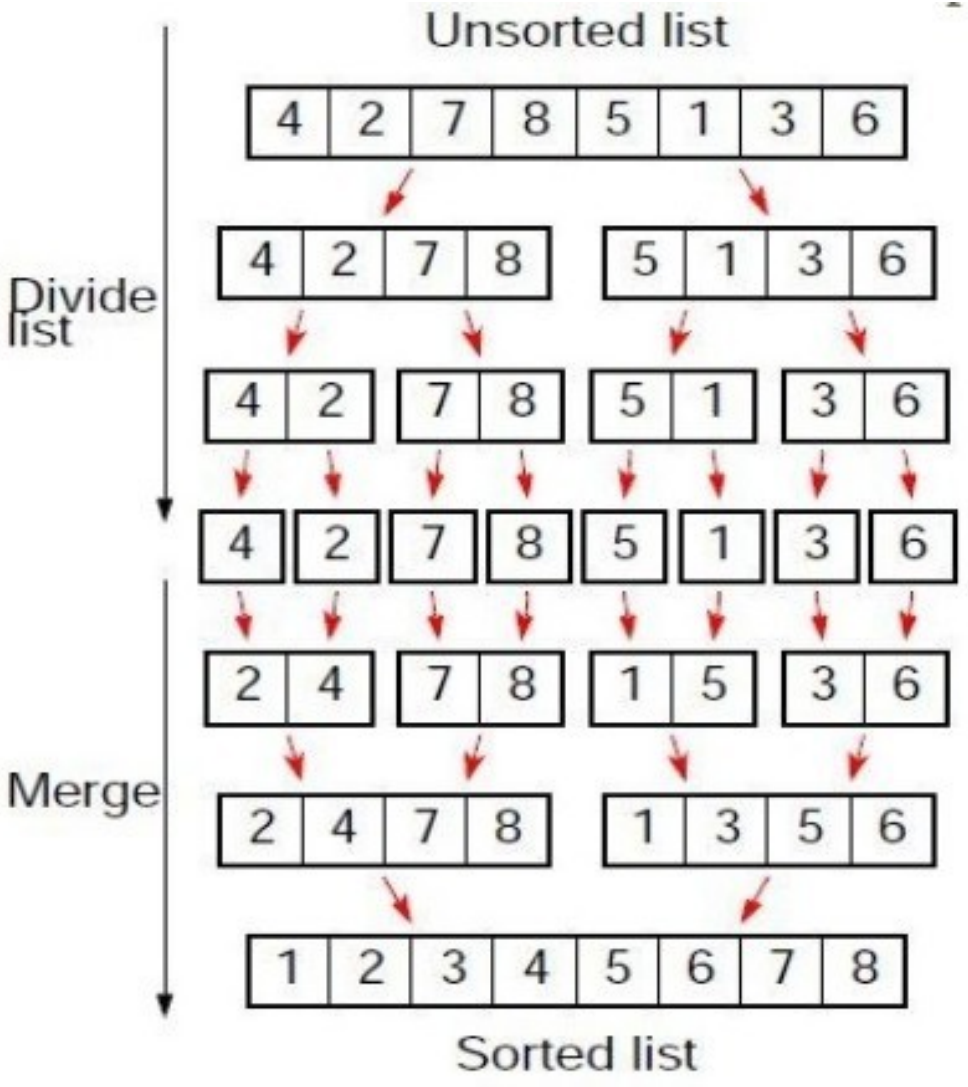
เมื่อได้รับภาระงานแล้ว แต่ละเธรดจะวนรอบตัวแปร j และ k ของตนเองตามประโยค **private(j, k)** ตามลำดับ เพื่อที่จะคำนวณได้อย่างอิสระจากเธรดอื่นๆ

8.3.2 การเรียงข้อมูล (Sorting) แบบขนาน

การเรียงข้อมูลเป็นอัลกอริทึมพื้นฐานที่ผู้อ่านควรจะได้เรียนในวิชา เช่น วิชา Algorithm Analysis and Design หรือวิชา Data Structure and Algorithm เป็นต้น อัลกอริทึมเรียงข้อมูลเริ่มต้นจากการทำงานแบบอนุกรม และพัฒนาให้เป็นอัลกอริทึมแบบขนานเมื่อฮาร์ดแวร์รองรับ N คือ ขนาดของข้อมูลที่ต้องการเรียงลำดับ ความซับซ้อนเฉลี่ยในรูปของพีชคณิต $O(\cdot)$ ขึ้นกับอัลกอริทึมที่เลือก ยกตัวอย่าง เช่น

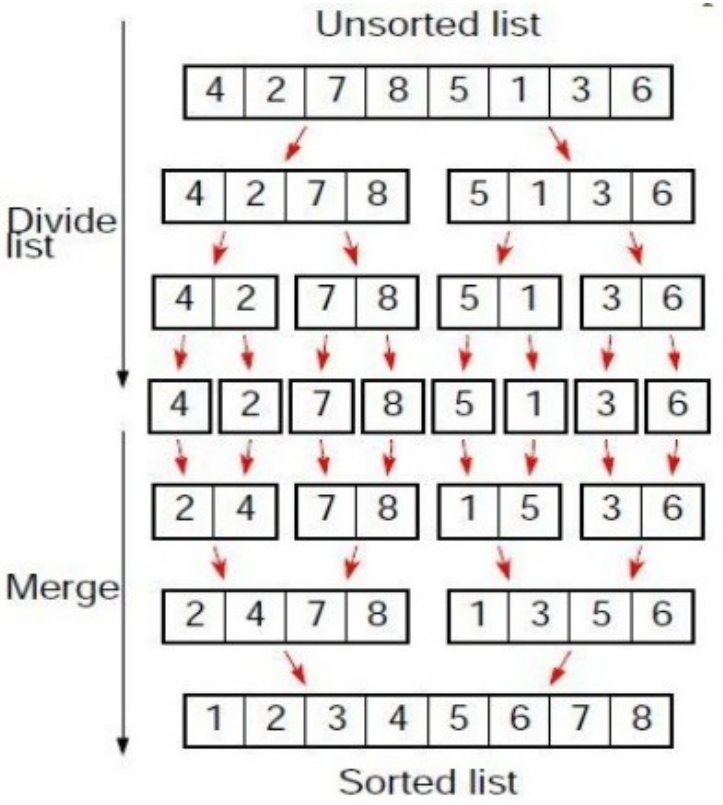
- การเรียงข้อมูลพื้นฐาน เช่น อัลกอริทึม Bubble Sort มีความซับซ้อนเฉลี่ยเท่ากับ $O(N^2)$ สามารถใช้ได้กับข้อมูลทุกประเภท
- การเรียงข้อมูลตามค่าประจำตำแหน่ง เช่น Radix Sort เหมาะกับข้อมูลที่เป็นเลขจำนวนเต็ม
- การเรียงข้อมูลด้วยหลักการแบ่งและยึดครอง (Divide and Conquer) เช่น MergeSort และ Quick-Sort มีความซับซ้อนเฉลี่ยเป็น $O(N \log_2 N)$ เหมาะกับข้อมูลทุกประเภท

8.3.2 การเรียงข้อมูล (Sorting) แบบขนาน



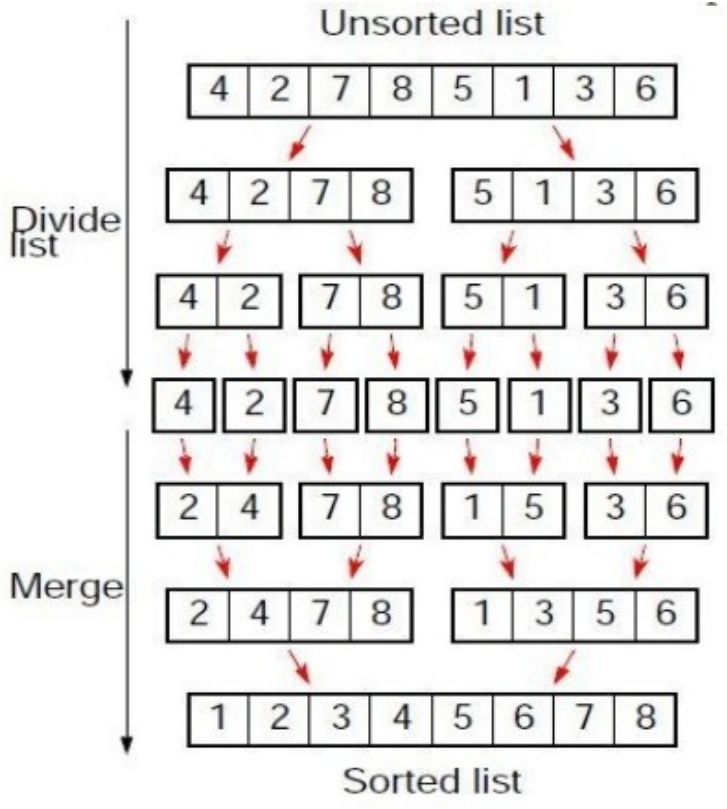
8.3.2 การเรียงข้อมูล (Sorting) แบบขนาน

```
void mergesort_omp(int a[], int size, int temp[], int threads) {
  if ( threads == 1) {
    mergesort_serial(a, size, temp);
  }
  else if (threads > 1) {
    #pragma omp parallel sections
    {
      #pragma omp section
      mergesort_omp(a, size/2, temp, threads/2);
      #pragma omp section
      mergesort_omp(a+size/2,size-size/2,temp+size/2,threads-threads/2);
    }
    merge(a, size, temp);
  } // threads > 1
}
```



8.3.2 การเรียงข้อมูล (Sorting) แบบขนาน

```
void mergesort_serial(int a[], int size, int temp[]) {
  int i;
  if (size == 2) {
    if (a[0] <= a[1])
      return;
    else {
      SWAP(a[0], a[1]);
      return;
    }
  }
  mergesort_serial(a, size/2, temp);
  mergesort_serial(a + size/2, size - size/2, temp);
  merge(a, size, temp);
}
```



สรุปท้ายบท

การคำนวณแบบอนุกรมเป็นพื้นฐานของขบวนการคิดของมนุษย์ แต่ความต้องการของคอมพิวเตอร์ที่หลากหลาย และปริมาณข้อมูลมากขึ้นเรื่อยๆ ทำให้วิวัฒนาการของคอมพิวเตอร์โน้มเอนมาเพื่อรองรับการคำนวณแบบขนานมากขึ้น คอมพิวเตอร์แบบขนานมี 2 ชนิดแบ่งตามการใช้งานหน่วยความจำหลัก คือ ชนิดมัลติคอร์และชนิดแมนีคอร์ซึ่งใช้หน่วยความจำร่วมกัน และชนิดมัลติคอมพิวเตอร์ซึ่งมีหน่วยความจำอิสระจากกัน การพัฒนาซอฟต์แวร์แบบขนานด้วยไลบรารี OpenMP ซึ่งเป็นที่ยอมรับและนิยมทั่วโลกรวมถึงซูเปอร์คอมพิวเตอร์ Fugaku สามารถรองรับความขนานของข้อมูล ความขนานของงานย่อย และความขนานลูกผสม

References

- https://royalsocietypublishing.org/doi/10.1098/rsta.2019.0061
- Micron Technology, I. (2004). Nand flash memory: Mt29f2g08aabwp/mt29f2g16aabwp/mt29f4g08babwp/mt29f4g16babwp/mt29f8g08fabwp.
- https://www.raspberrypi.org/forums/viewtopic.php?t=63750
- https://www.researchgate.net/figure/Block-Diagram-of-Micro-SD-card_fig6_306236972
- https://gabrieletolomei.wordpress.com/miscellanea/operating-systems/in-memory-layout/
- https://freedompenguin.com/articles/how-to/learning-the-linux-file-system
- Harris, D. and S. Harris (2013). Digital Design and Computer Architecture (1st ed.). USA: Morgan Kauffman Publishing.
- https://learn.adafruit.com/resizing-raspberry-pi-boot-partition/edit-partitions

References

- https://www.researchgate.net/figure/Block-Diagram-of-Micro-SD-card_fig6_306236972
- https://gabrieletolomei.wordpress.com/miscellanea/operating-systems/in-memory-layout/
- https://freedompenguin.com/articles/how-to/learning-the-linux-file-system
- Harris, D. and S. Harris (2013). Digital Design and Computer Architecture (1st ed.). USA: Morgan Kauffman Publishing.
- https://learn.adafruit.com/resizing-raspberry-pi-boot-partition/edit-partitions
- https://medium.com/ai%C2%B3-theory-practice-business/fastai-partii-lesson08-notes-fddcdb6526bb
- https://www.semanticscholar.org/paper/Multi-Core-Programming-Increasing-Performance-Roberts/0efa17a35457f6ad00a4834d9e2a20339191598b