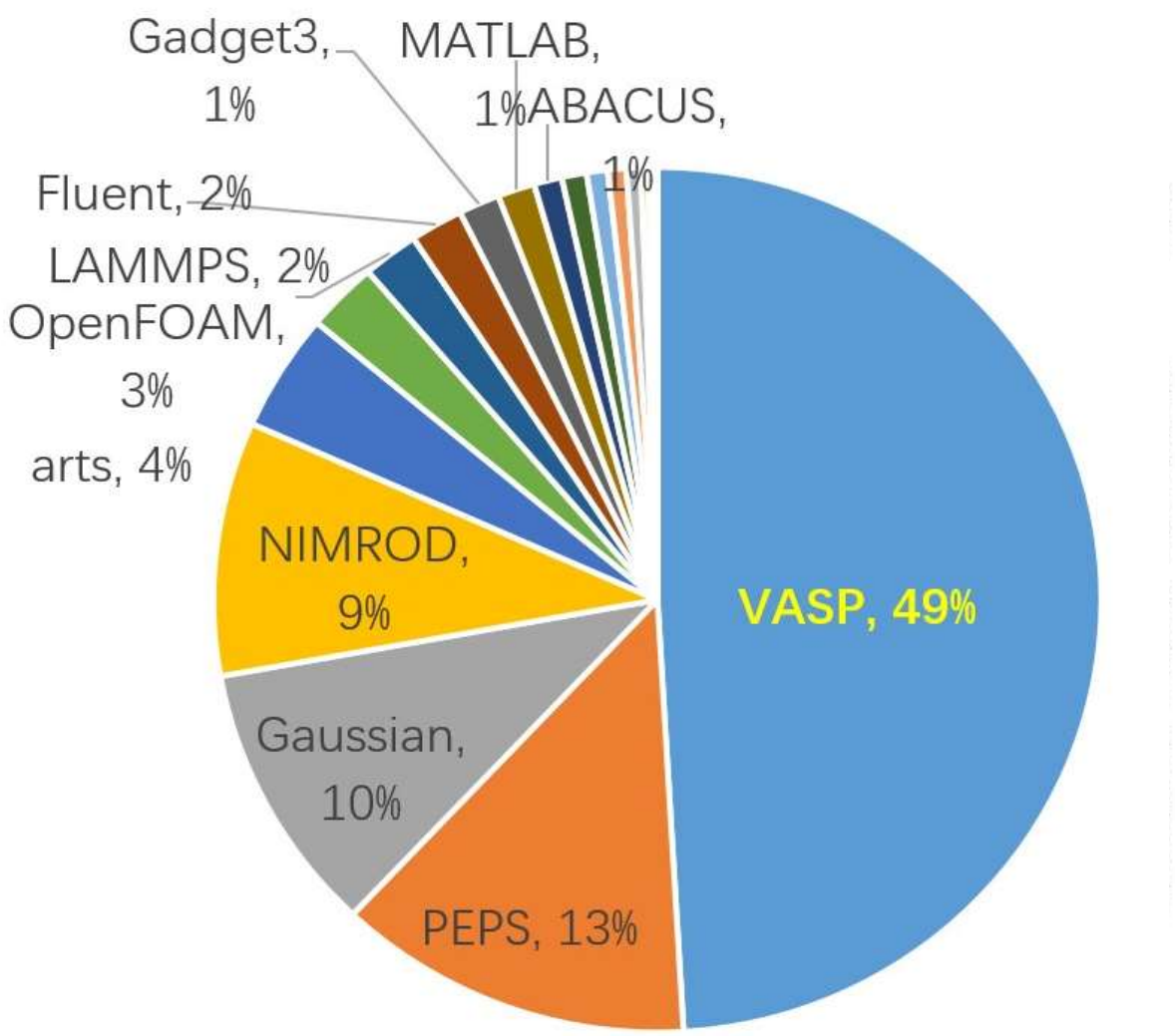
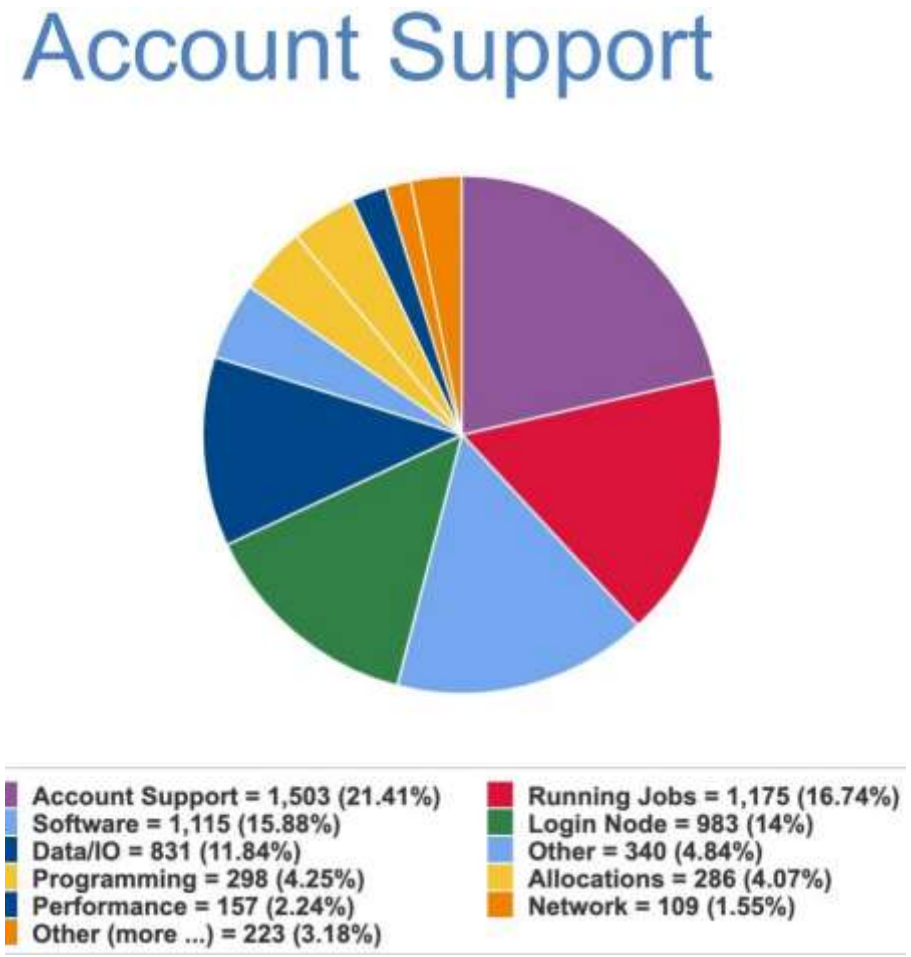
VASP应用的大规模并行计算与运行优化

张文帅

2023年5月30日

VASP应用与用户背景



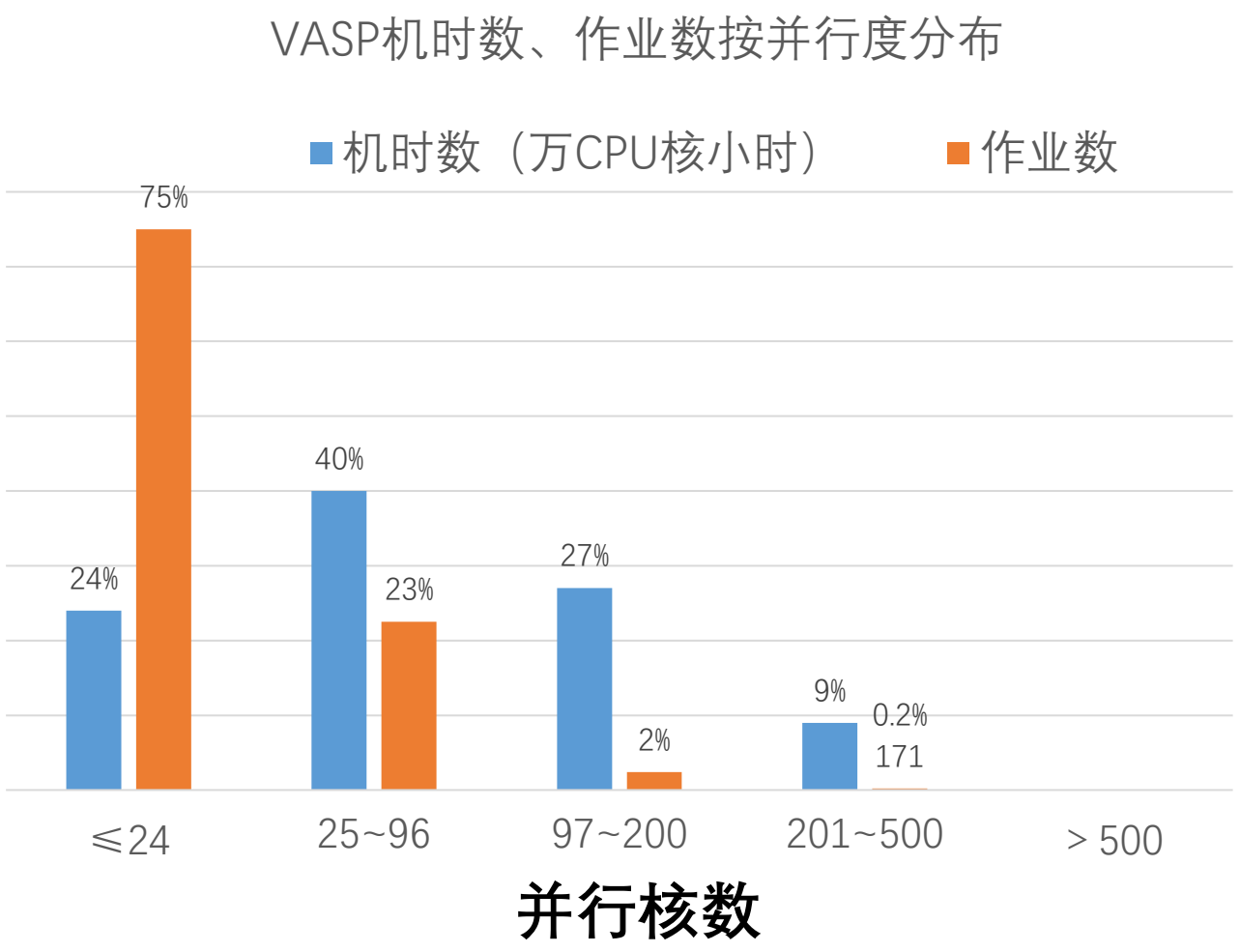


VASP作业运行问题

并行规模一般，25-200核左右作业占据了近70%机时，使用并行度不高。

为什么？
怎么办？

- 核数多的作业排队长：解决资源问题
- 部分土豪用户/学生，滥用资源，且效率不高，加剧资源不足：解决利用效率。
- 核数多时加速效果不好：提高计算速度



- 高性能计算软硬件技术背景
- 运行现状
- 优化之术

软件并行算法可扩展性

- 强可扩展性：固定问题规模，改变计算核心使用量

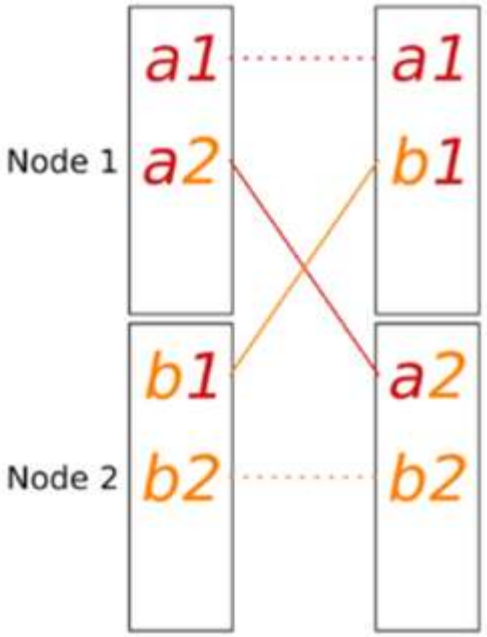
  Time(N) = Time(1) / N

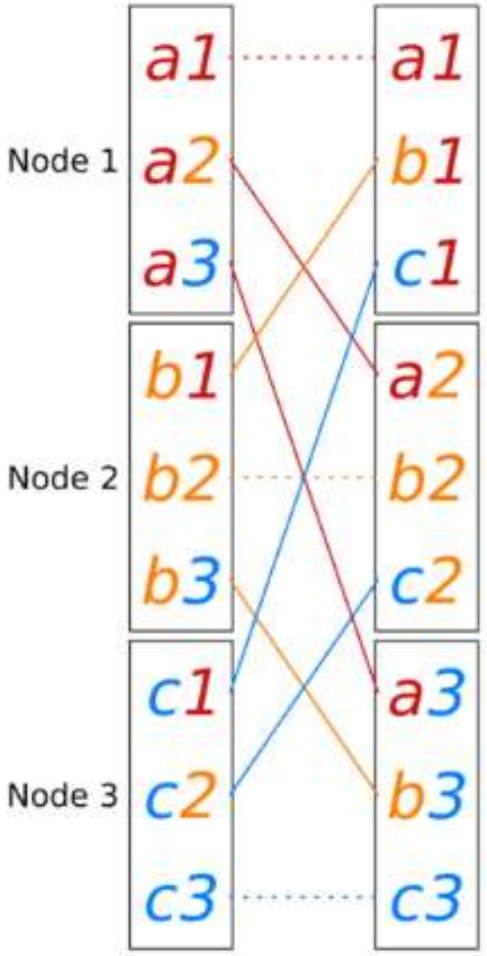
  N为CPU处理核心数

- 弱可扩展性：扩大问题规模，相应增加计算核心

  Workload(1)/1 ~? Workload(M)/N

弱可扩展性：扩大问题规模，相应增加计算核心





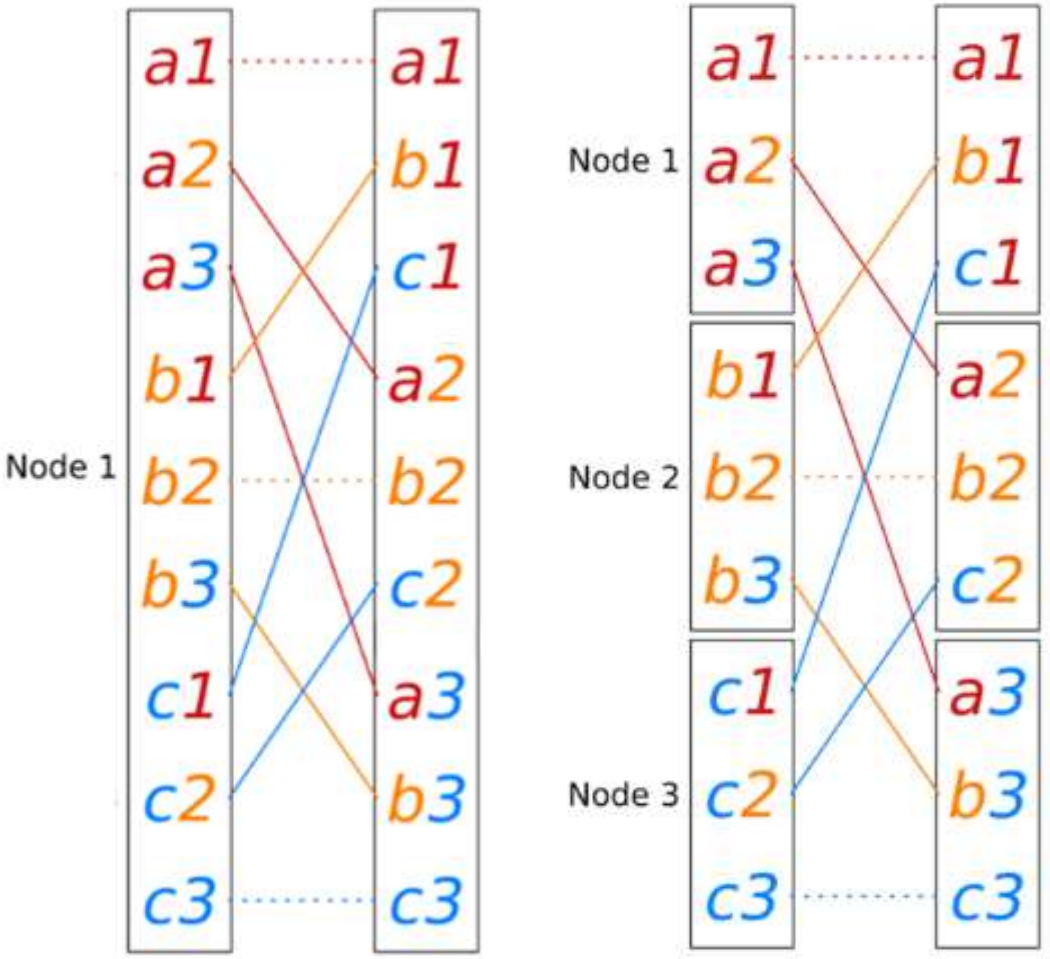
使用资源数Nodes: 2->3，任务数4->9

数据末端b2 -> c3，任务数~o(N^2):

每个计算核心需要与其他所有计算核心通讯，每核心的通讯任务数：1 -> 2。

All to All 数据操作

强可扩展性：固定问题规模，改变计算核心使用量



All to All 数据操作

使用资源数Nodes： 1->3：

1. 同一个核心内处理时：可在L1/L2缓存内操作
2. 当多核心并行处理时：每核心取数据的任务数降低N倍，但是时间并不一定降低N倍，因为缓存内操作变为核间通讯，继续增大并行计算核心将触发跨节点通讯。

此外，可并行任务不够多，无法多于6组并行

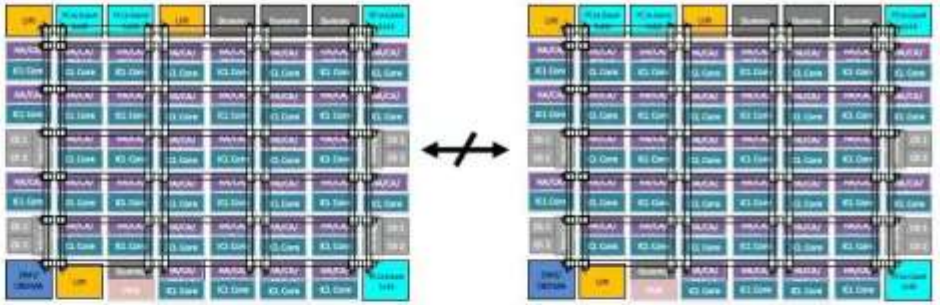
- 硬件发展趋势

3rd Gen Intel Xeon Scalable Processors

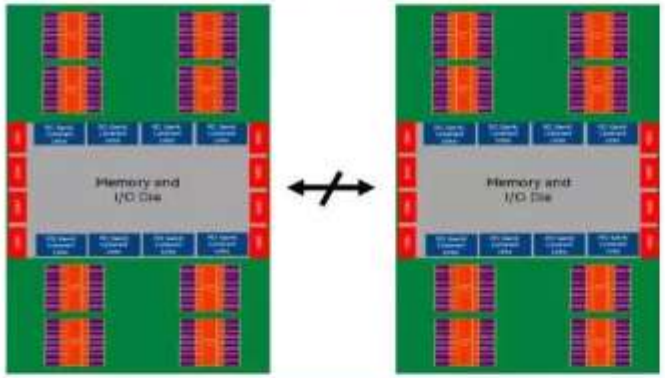
Processor architecture, cache latencies

访存延迟差异巨大

3rd Gen Intel Xeon Scalable Processors



3rd Gen AMD EPYC



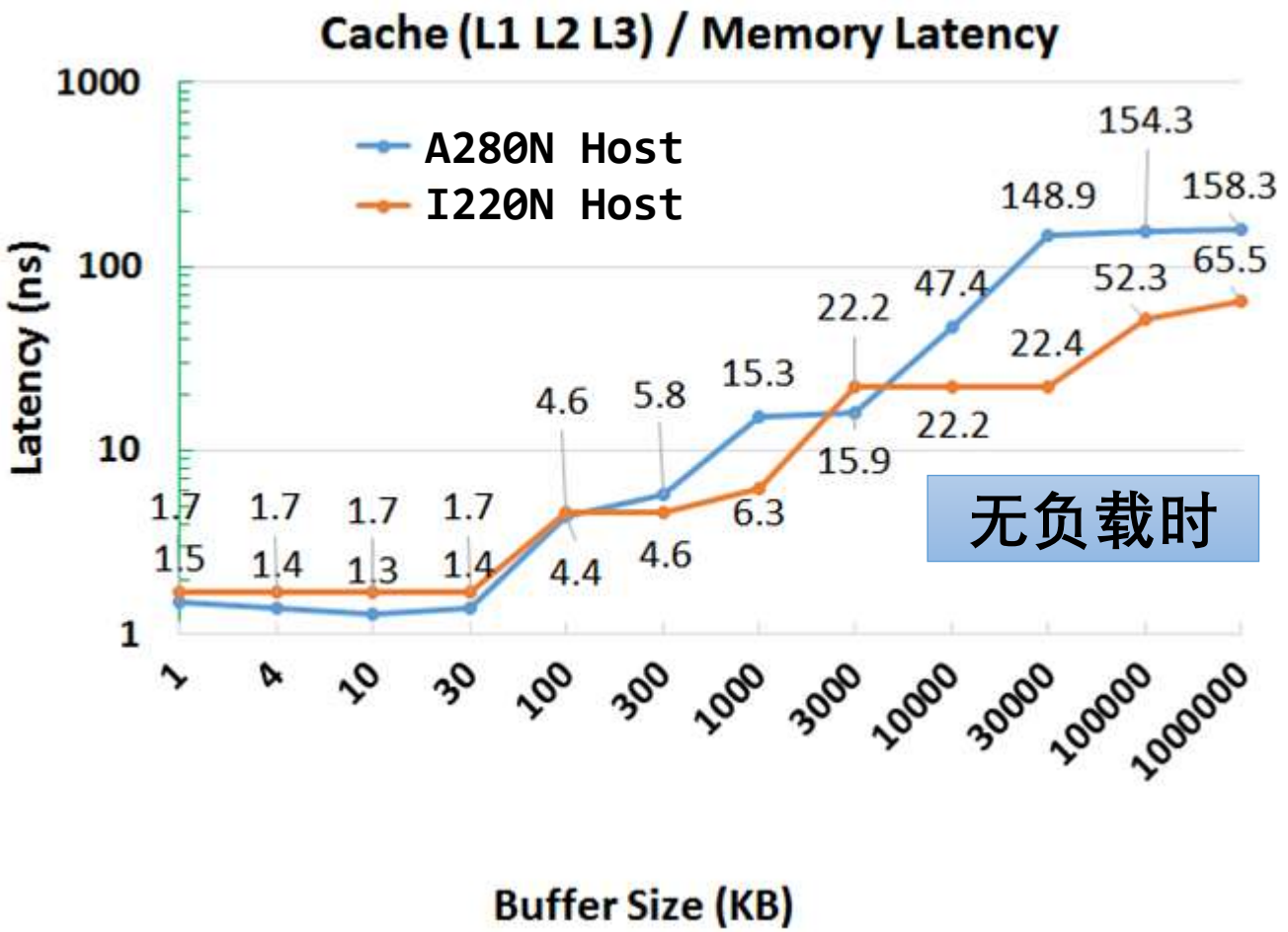
Latency	Intel Xeon Platinum 8380 Processor (Ice Lake)	AMD EPYC 7763 Processor (Milan)	Intel Xeon Platinum 8280 Processor (Cascade Lake)
L1 hit cache, cycles	5	4	4
L2 hit cache, cycles	14	12	14
L3 hit cache (same socket), ns	21.7	13.4 (< 32MB) local die 112 (> 32MB) remote die	20.2
L3 hit cache (remote socket), ns	118	209	180

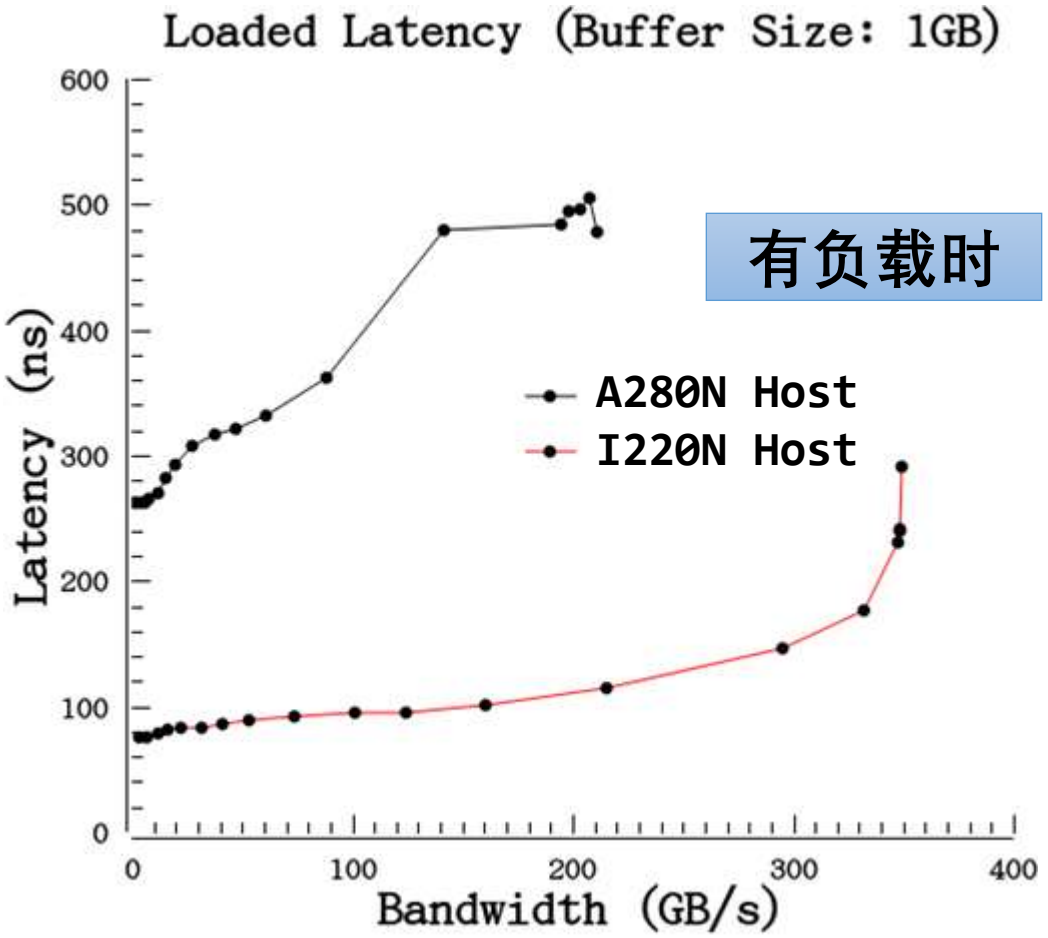
3rd Gen Intel Xeon Scalable processor have consistent and low latencies to local cache and the second socket

CPU缓存、内存的延时与带宽测试

Intel(R) Memory Latency Checker v3.9

单机内读取数据延迟时间范围： 1.3 ns -> 500 ns



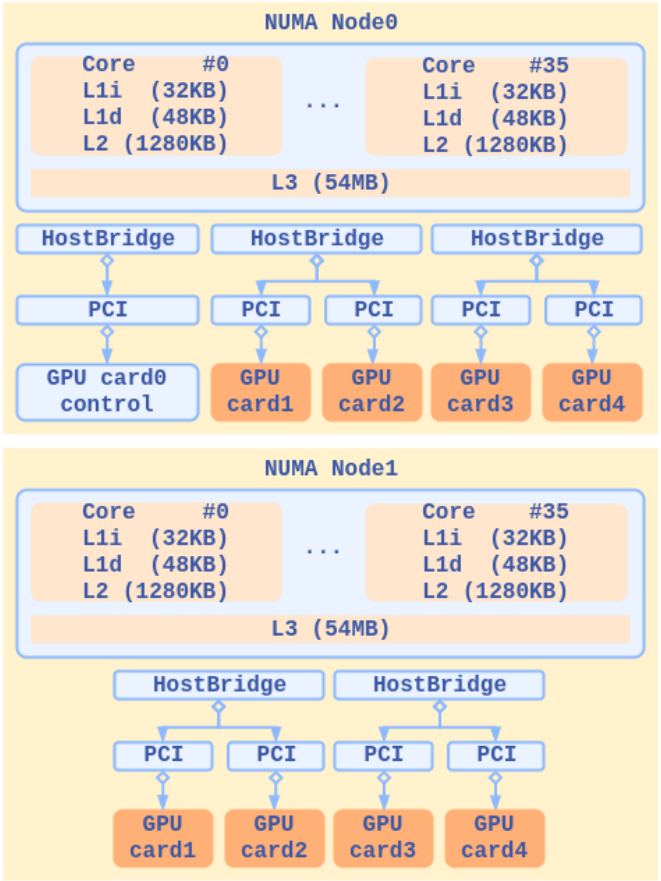


- A280NHost相比I220NHost具有稍低的L1 Cache延时(~ 1.4ns vs. 1.7ns)，具有相似的L2 Cache延时。
- I220NHost具有明显更小的内存(Memory)延时(~ 60ns vs. 150ns)。

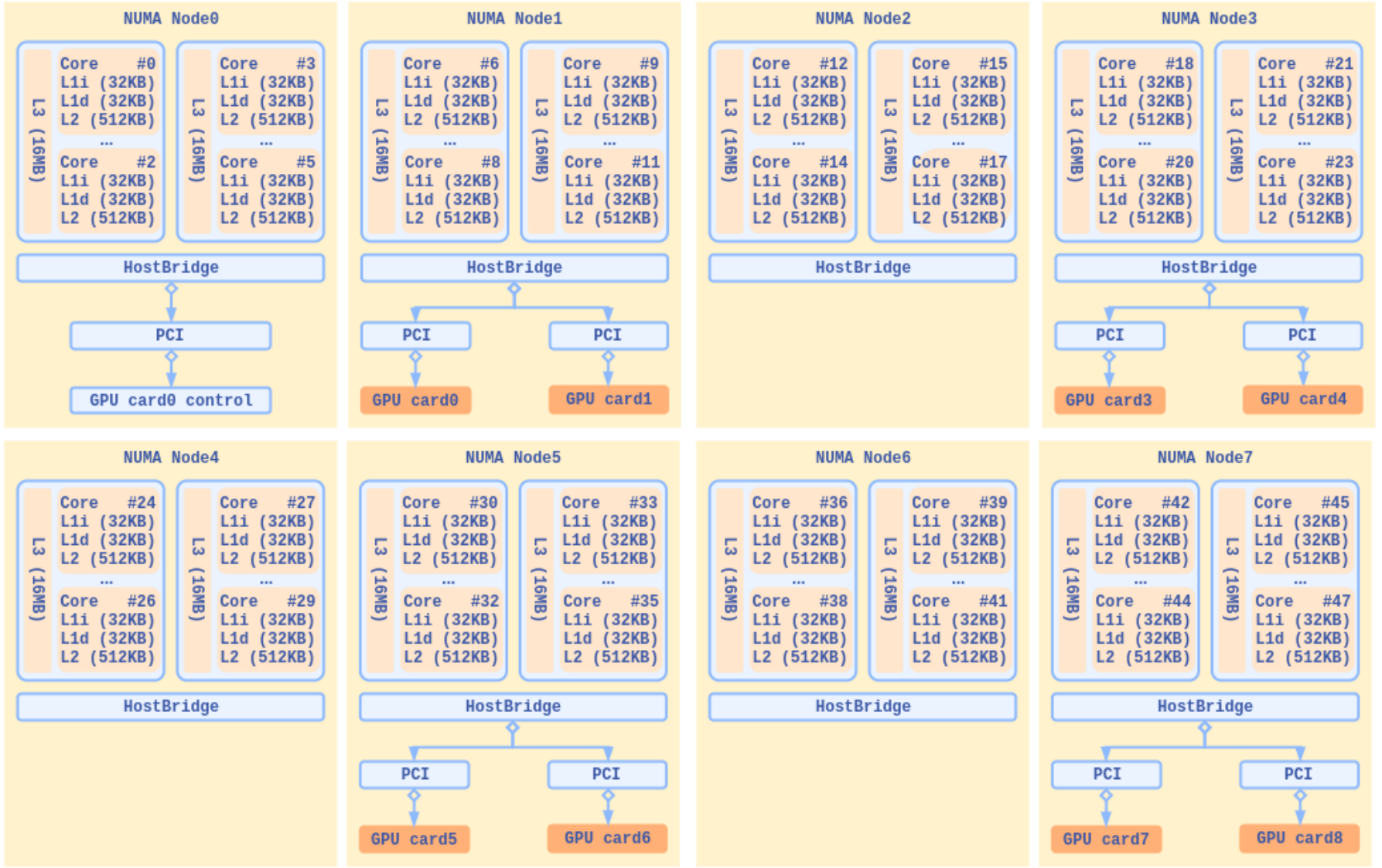
- A280NHost 最高负载仅能达到211GB/s，负载下的延时为500ns
- I220NHost最高负载可达到350GB/s，且该负载下的延时为290ns。

硬件趋势：更多的异构(NUMA分区等)

Intel CPU 2.20GHz （I220N）



AMD CPU 280GHz （A280N）



A280N Host具有8个NUMA分区，16个独立的L3 Cache；
I220N Host具有2个NUMA分区，2个独立的L3 Cache。

多节点间数据传输延迟与带宽

本地网络延迟 ~ 2 us

每列数据的涵义依次为
1、发送的单个数据包的大小，从0到16MB左右；
2、重复测量次数；
3、p2p通信延时，描述单个数据包往返总时间；
4、数据的流量带宽，描述每秒可发送的数据大小总量，也就是网速。

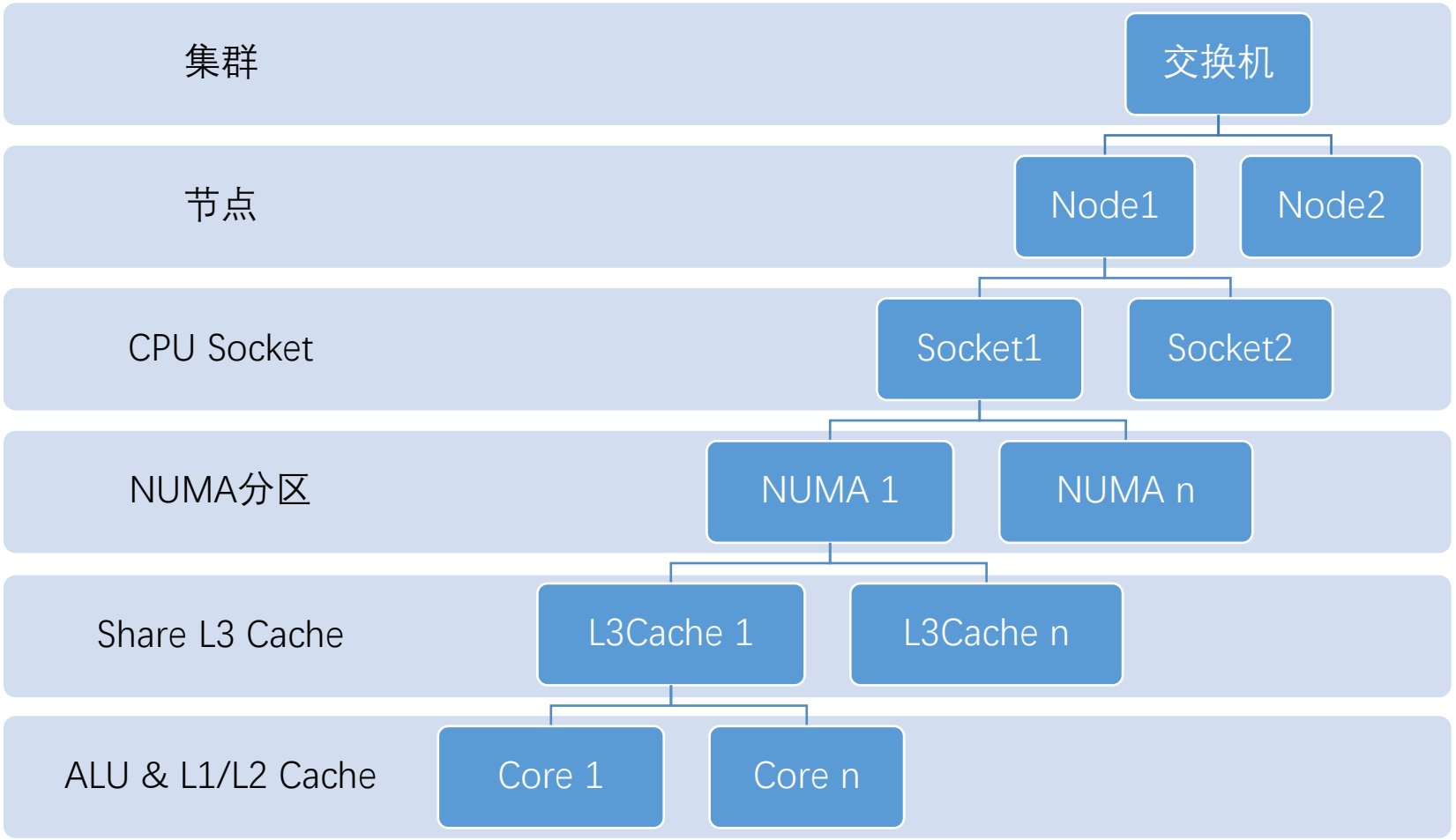
观察可知：
0 Byte数据包下，p2p延迟是2us。在数据包大小在1B-128B之间变化时，我们的每个数据包p2p往返的时间是几乎相同的，都约为2.4us，每秒可处理的消息数是相同的。在此范围内，当数据包大小倍增时，我们得到的网络带宽也相应倍增。（**极限压缩数据包并不总会改进通讯的效率，并不一定带来网络处理能力的提升**）

考虑100Gbps计算能力的网卡，它可以在1秒内处理$10^{10}/8$ Byte的数据，也就是在1us时间内处理1250 Byte数据，即在0.1us时间内处理完125 Byte数据。

```
#    Intel(R) MPI Benchmarks 2019 Update 4, MPI-1 part
#------------------------------------------------------------
# Date                  : Fri Jun 25 20:59:37 2021
# Machine               : x86_64
# System                : Linux
# Release               : 3.10.0-1062.9.1.el7.x86_64
# Version               : #1 SMP Fri Dec 6 15:49:49 UTC 2019
# MPI Version           : 3.1
# IMB-MPI1 PingPong -msglen len.dat -multi 1 -map 2x4
#------------------------------------------------------------
# Benchmarking Multi-PingPong
# ( 4 groups of 2 processes each running simultaneous )
# Group  0:     0    2
# Group  1:     1    3
# Group  2:     4    6
# Group  3:     5    7
#------------------------------------------------------------
 #bytes #repetitions      t[usec]   Mbytes/sec
      0         1000         2.01         0.00
      1         1000         2.38         0.42
      2         1000         2.39         0.84
      4         1000         2.39         1.67
      8         1000         2.39         3.34
     16         1000         2.35         6.80
     32         1000         2.40        13.34
     64         1000         2.41        26.52
    128         1000         2.47        51.89
    256         1000         3.29        77.73
    512         1000         3.46       148.13
   1024         1000         3.87       264.27
   4096         1000         6.33       647.57
 262144          160       117.13      2238.14
4194304           10       878.96      4771.90
4194304           10       874.57      4795.86
```

硬件发展总结

为软件开发带来较多困难



GPU

FPGA

ASIC

…

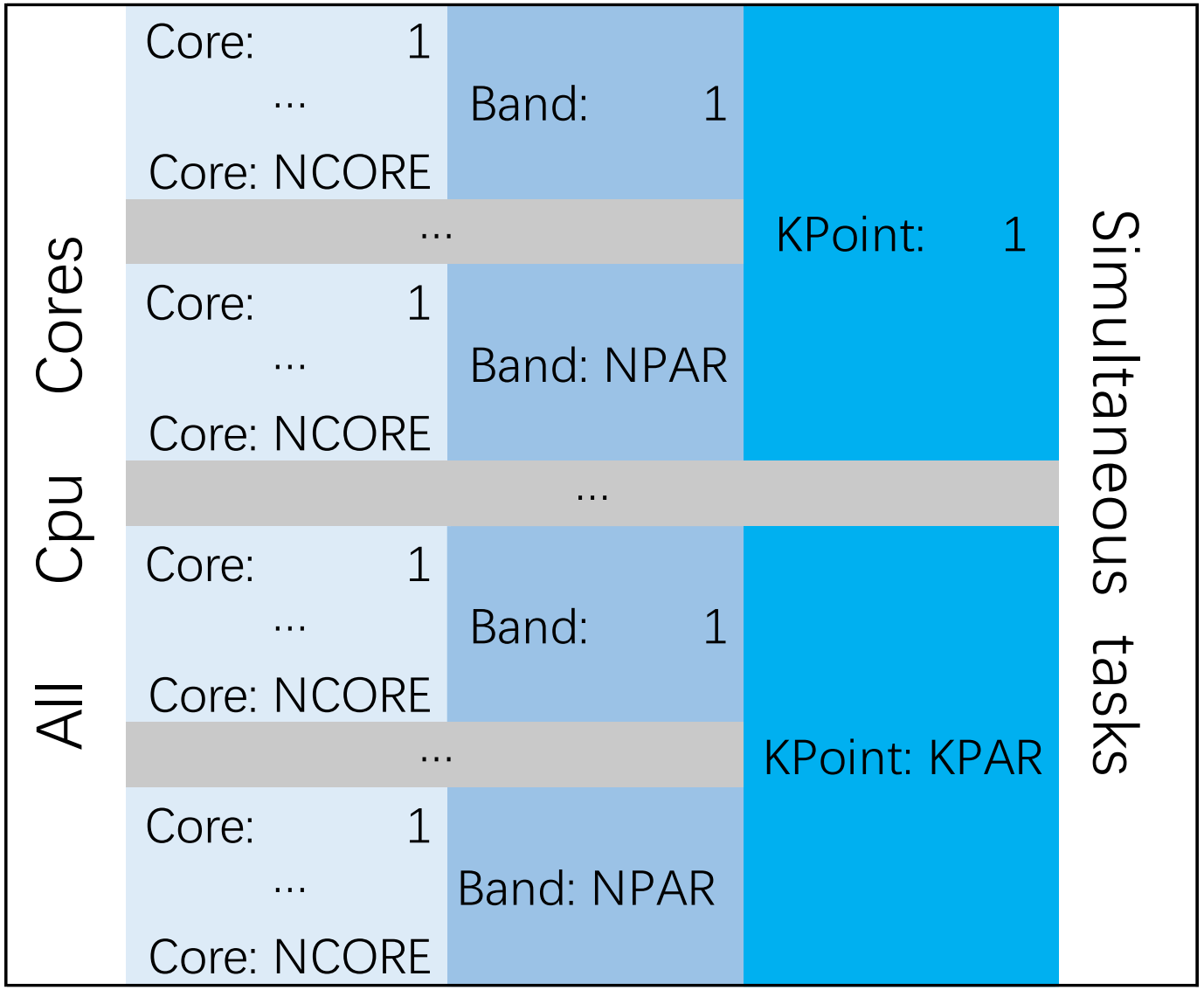
延迟：1.3ns -> 500ns -> 2us（时间~ **x 1000**）

数据：1 Byte -> 128 Byte -> 32K Byte -> 4MB Byte (空间~ **x 4,000,000**)

- 软件发展趋势

VASP应用发展

- ✓ **k** : K points 并行
- ✓ ***n*** : Band 并行
- ✓ **r** <-> PW : FFT 并行
- ✓ OpenMP并行



The Kohn-Sham equation:

$$\left(-\frac{1}{2}\Delta + V_{\text{ext}}(\mathbf{r}) + V_{\text{H}}(\mathbf{r}) + V_{\text{xc}}(\mathbf{r})\right)\psi_{n\mathbf{k}}(\mathbf{r}) = \epsilon_{n\mathbf{k}}\psi_{n\mathbf{k}}(\mathbf{r})$$

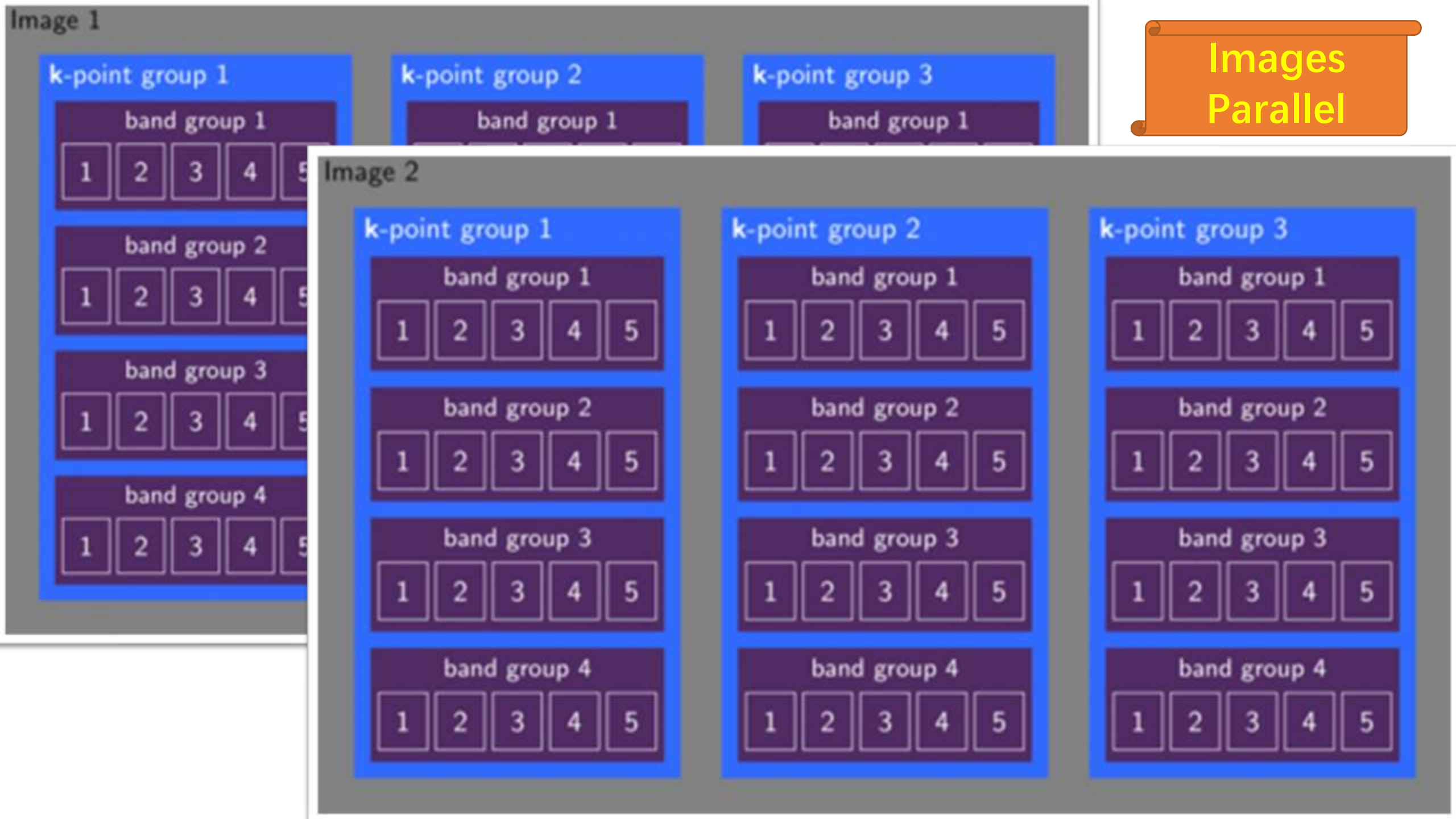
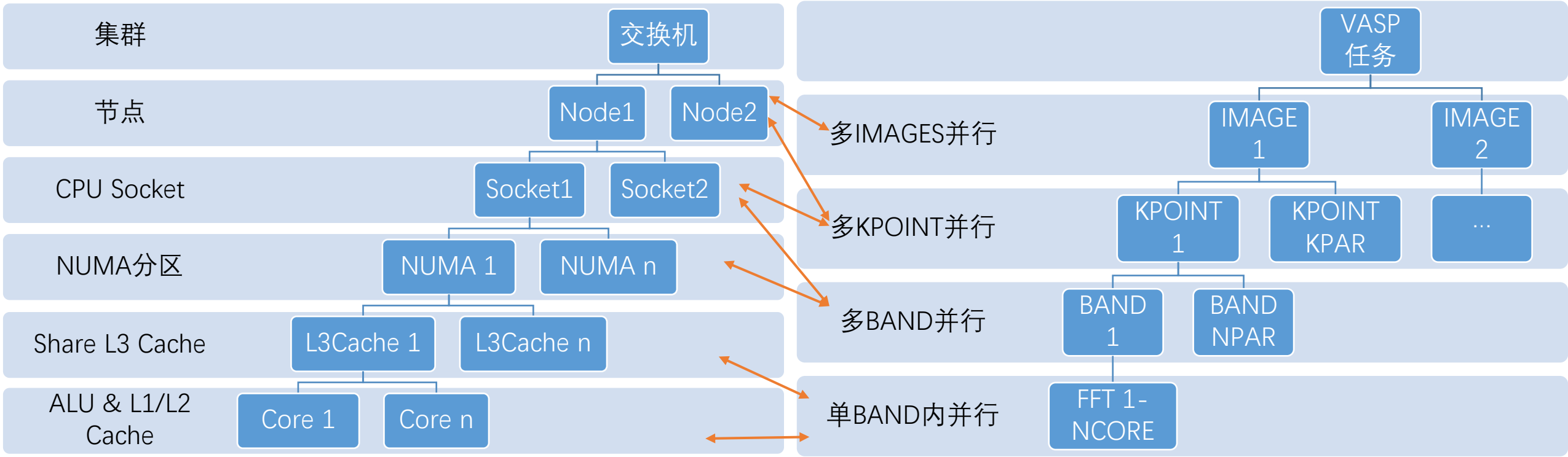


Image 1
k-point group 1
band group 1
1 2 3 4 5
band group 2
1 2 3 4 5
band group 3
1 2 3 4 5
band group 4
1 2 3 4 5
k-point group 2
band group 1
k-point group 3
band group 1
Image 2
k-point group 1
band group 1
1 2 3 4 5
band group 2
1 2 3 4 5
band group 3
1 2 3 4 5
band group 4
1 2 3 4 5
k-point group 2
band group 1
1 2 3 4 5
band group 2
1 2 3 4 5
band group 3
1 2 3 4 5
band group 4
1 2 3 4 5
k-point group 3
band group 1
1 2 3 4 5
band group 2
1 2 3 4 5
band group 3
1 2 3 4 5
band group 4
1 2 3 4 5
Images
Parallel

并行任务分发越发复杂


集群
交换机
节点
Node1
Node2
CPU Socket
Socket1
Socket2
NUMA分区
NUMA 1
NUMA n
Share L3 Cache
L3Cache 1
L3Cache n
ALU & L1/L2 Cache
Core 1
Core n
VASP 任务
多IMAGES并行
IMAGE 1
IMAGE 2
多KPOINT并行
KPOINT 1
KPOINT KPAR
...
多BAND并行
BAND 1
BAND NPAR
单BAND内并行
FFT 1-NCORE

并行运行的软硬件关联性越发复杂

- 每一级别具有各自特性的并行扩展能力，以及任务均衡所需的分配因子，其对应不同硬件级别的配置（核心数、NUMA分区数、Socket数量、节点数）

- 每一级别的并行化，可能产生不同的内存需求（K-points并行会导致内存使用量激增），由此产生跟硬件内存配置相关的并行参数限制

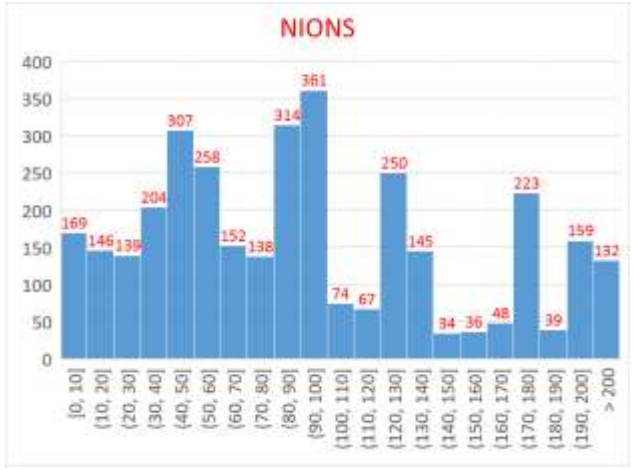
- 计算之道
- 运行现状
- 优化之术

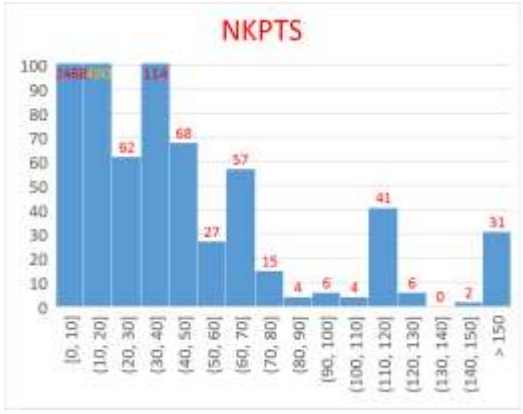
VASP作业特征

抽样统计3060个VASP计算作业，其中HF/DFT混合计算作业只有197个，占比6%；RPA计算接近为零。

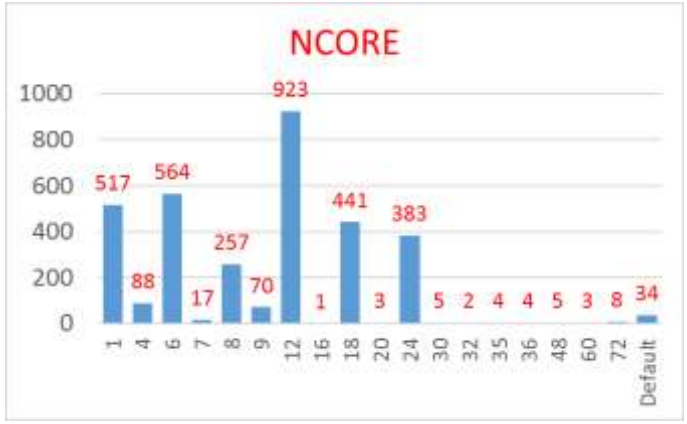
抽样统计3095个VASP计算作业，其中ISPIN=2有2784个，占比90%。

抽样统计3095个VASP计算作业，其中GammaOnly计算有491个，占比16%。





大量使用默认并行参数



一些公开的参数建议失效

建议1：NPAR = 4 ~ approx SQRT(number of cores)

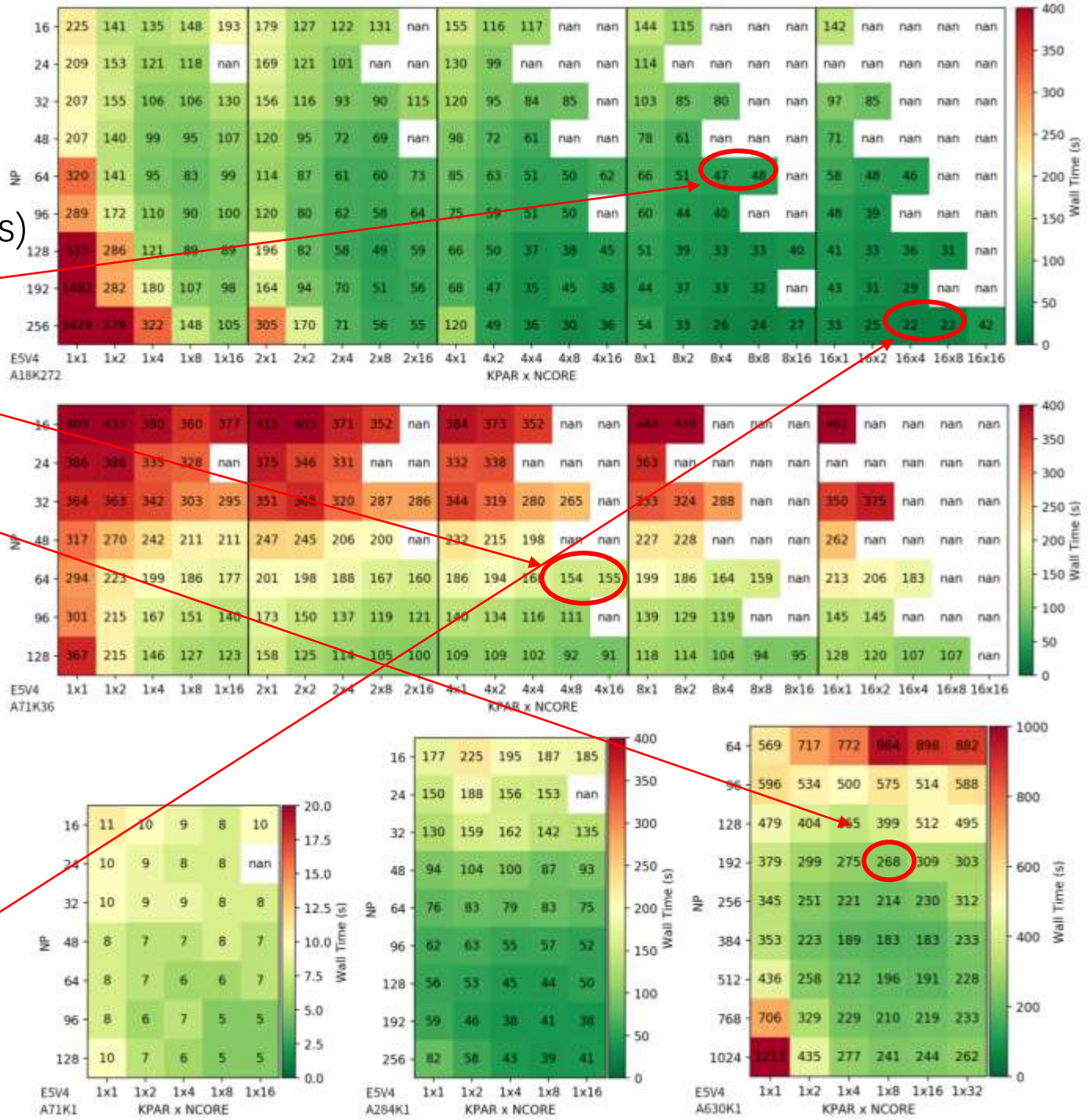
建议NPAR = 4~8，实际最优 NPAR= 1 或 2

建议NPAR = 4~14，实际最优 NPAR= 24

建议2：NPAR = number of cores per compute node

建议3：**not recommend** attempting run with **KPAR>compute nodes**, even though you may have more k-points than compute nodes.

建议KPAR < 10，实际最优：KPAR= 16

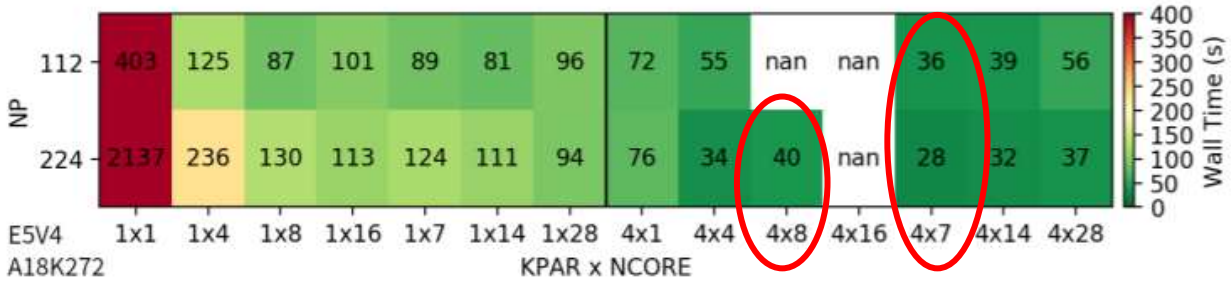


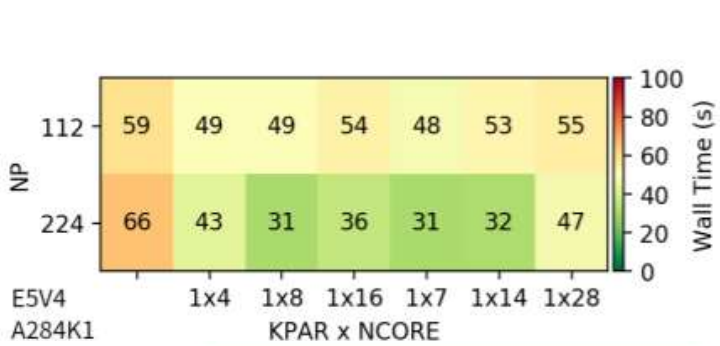
初步的测试探索

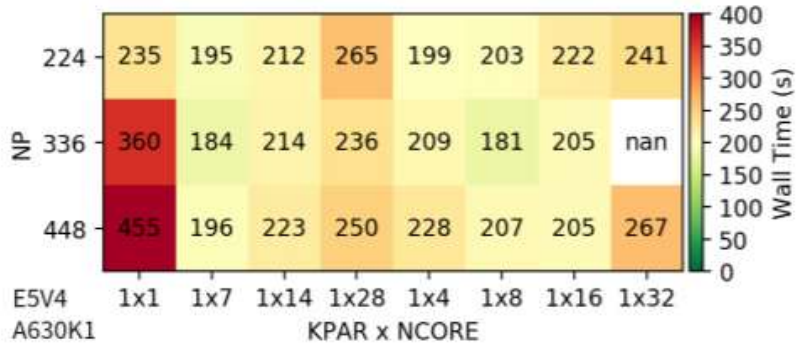
强硬件依赖性

单节点核心数可被 NCORE 整除时更优。

应测试，可使 MPI通讯耗时由199/254 降低至 131/182







254.23s
Elapsed Time

107.76
SP FLOPS

0.89
CPI
(MAX 0.93, MIN 0.66)

Your application is MPI bound. This may be caused by high busy wait time inside the library (imbalance), non-optimal communication schema or MPI library settings. Use MPI profiling tools like Intel® Trace Analyzer and Collector to explore performance bottlenecks.

	Current run	Target	Delta
MPI Time	78.53%	<15%	
Memory Stalls	22.23%	<20%	
FPU Utilization	2.31%	>50%	
I/O Bound	0.26%	<10%	

MPI Time
78.53% of Elapsed Time
(199.64s)
MPI Imbalance
26.37% of Elapsed Time
(67.04s)

Memory Footprint
MEAN 146.84 MB, PEAK 156.21 MB

TOP 5 MPI functions

Func	%
Bcast	36.49
Allreduce	22.17
Alltoall	14.89
Barrier	14.55
Alltoallv	8.41

Memory Stalls
22.23% of pipeline slots
Cache Stalls
20.58% of cycles
DRAM Stalls
1.03% of cycles
NUMA
18.87% of remote accesses

FPU Utilization
2.31%
SP FLOPs per Cycle
0.74 Out of 32.00
Vector Capacity Usage
83.19%
FP Instruction Mix
% of Packed FP Instr.: 94.34%
% of 128-bit: 25.09%
% of 256-bit: 69.26%
% of Scalar FP Instr.: 5.66%
FP Arith/Mem Rd Instr. Ratio
0.26
FP Arith/Mem Wr Instr. Ratio
1.06

I/O Bound
0.26%
(MEAN 37.19, PEAK 1.15)
Read
MEAN 1.2 MB, MAX 1.2 MB
Write
MEAN 4.0 MB, MAX 6.7 MB

182.97s
Elapsed Time

135.82
SP FLOPS

0.90
CPI
(MAX 0.95, MIN 0.70)

Your application is MPI bound. This may be caused by high busy wait time inside the library (imbalance), non-optimal communication schema or MPI library settings. Use MPI profiling tools like Intel® Trace Analyzer and Collector to explore performance bottlenecks.

	Current run	Target	Delta
MPI Time	71.89%	<15%	
Memory Stalls	23.82%	<20%	
FPU Utilization	2.92%	>50%	
I/O Bound	0.08%	<10%	

MPI Time
71.89% of Elapsed Time
(131.54s)
MPI Imbalance
19.44% of Elapsed Time
(35.57s)

Memory Footprint
MEAN 144.65 MB, PEAK 153.27 MB

TOP 5 MPI functions

Func	%
Bcast	41.67
Allreduce	21.97
Alltoall	14.35
Barrier	10.37
Alltoallv	6.03

Memory Stalls
23.82% of pipeline slots
Cache Stalls
22.10% of cycles
DRAM Stalls
0.94% of cycles
NUMA
16.66% of remote accesses

FPU Utilization
2.92%
SP FLOPs per Cycle
0.93 Out of 32.00
Vector Capacity Usage
82.02%
FP Instruction Mix
% of Packed FP Instr.: 93.87%
% of 128-bit: 26.74%
% of 256-bit: 67.14%
% of Scalar FP Instr.: 6.13%
FP Arith/Mem Rd Instr. Ratio
0.34
FP Arith/Mem Wr Instr. Ratio
1.34

I/O Bound
0.08%
(MEAN 7.86, PEAK 0.29)
Read
MEAN 1.2 MB, MAX 1.2 MB
Write
MEAN 4.0 MB, MAX 6.7 MB

56核 -> 112核

KPAR: 1->2

NCORE: 8->7

254.23s

Elapsed Time

-> 411 s

-> 173 s

-> 90 s

Your application is MPI bound. This may be caused by high busy wait time inside the library (imbalance), non-optimal communication schema or MPI library settings. Use MPI profiling tools like Intel® Trace Analyzer and Collector to explore performance bottlenecks.

	Current run	Target	Delta
MPI Time	78.53%	<15%	
Memory Stalls	22.23%	<20%	
FPU Utilization	2.31%	>50%	
I/O Bound	0.26%	<10%	

107.76

SP FLOPS

0.89

CPI

(MAX 0.93, MIN 0.66)

-> 359 s

-> 143 s

-> 68 s

MPI Time

78.53% of Elapsed Time

(199.64s)

MPI Imbalance

26.37% of Elapsed Time

(67.04s)

Memory Footprint

MEAN 146.84 MB, PEAK 156.21 MB

TOP 5 MPI functions

Func	%
Bcast	36.49
Allreduce	22.17
Alltoall	14.89
Barrier	14.55
Alltoallv	8.41

Memory Stalls

22.23% of pipeline slots

Cache Stalls

20.58% of cycles

DRAM Stalls

1.03% of cycles

NUMA

18.87% of remote accesses

FPU Utilization

2.31%

SP FLOPs per Cycle

0.74 Out of 32.00

Vector Capacity Usage

83.19%

FP Instruction Mix

% of Packed FP Instr.: 94.34%

% of 128-bit: 25.09%

% of 256-bit: 69.26%

% of Scalar FP Instr.: 5.66%

Application: *vasp_std*
Report creation date: *2023-01-25 12:18:18*
Number of ranks: *112*
Ranks per node: *28*
HW Platform: *Intel(R) Xeon(R) Processor code named Broadwell*
Logical Core Count per node: *28*
Collector type: *Driverless Perf per-proce*

默认参数改为 2 x 2 x 28:
-> 513 s

2715.37s
Elapsed Time

Your application is memory bound.

Use memory access analysis tools like Intel® VTune™ Amplifier for a detailed metric breakdown by memory hierarchy, memory bandwidth, and correlation by memory objects.

	Current run	Target	Delta
MPI Time	9.10%	<10%	
Memory Stalls	69.15%	<20%	
FPU Utilization	4.47%	>50%	
I/O Bound	0.00%	<10%	

430.92
SP GFLOPS

2.38
CPI
(MAX 2.82, MIN 1.04)

-> 172 s 33 %

-> 33 %

-> 9.5 %

MPI Time
247.05s
9.10% of Elapsed Time

MPI Imbalance
211.14s
7.78% of Elapsed Time

TOP 5 MPI Functions	%
Alltoall	3.25
Allreduce	2.74
Bcast	2.42
Barrier	0.44
Alltoallv	0.19

Memory Stalls
69.15% of pipeline slots

Cache Stalls
23.55% of cycles

DRAM Stalls
44.80% of cycles

DRAM Bandwidth
Not Available

NUMA
0.49% of remote accesses

FPU Utilization
4.47%

SP FLOPs per Cycle
1.43 Out of 32.00

Vector Capacity Usage
94.56%

FP Instruction Mix
% of Packed FP Instr.: 99.17%
% of 128-bit: 9.66%
% of 256-bit: 89.52%
% of Scalar FP Instr.: 0.82%

FP Arith/Mem Rd Instr. Ratio

I/O Bound
0.00%
(AVG 0.05, PEAK 0.10)

Read
AVG 1014.9 KB, MAX 1.0 MB

Write
AVG 4.6 KB, MAX 323.1 KB

需针对具体作业做具体的个性化测试：自动化测试

"For the NPAR, I recommend doing a test to find out the most efficient number. e.g. run a same calculations multiple times with different NPAR. Also, do the same for LPLANE parameter as well. The manual instructs to use the number of node as NPAR as each parallel calculation can be run at each node minimizing communication overhead between each node. If not optimized, VASP takes extra time to comminucate between nodes, eating up your computation time. However, I have found that this instruction does not always hold up, and, really, this parameter is heavily dependent on the batch server/ node configuration. So, ***it is wise to do your own test to optimize this parameter (and other parameters as well).*** *"*

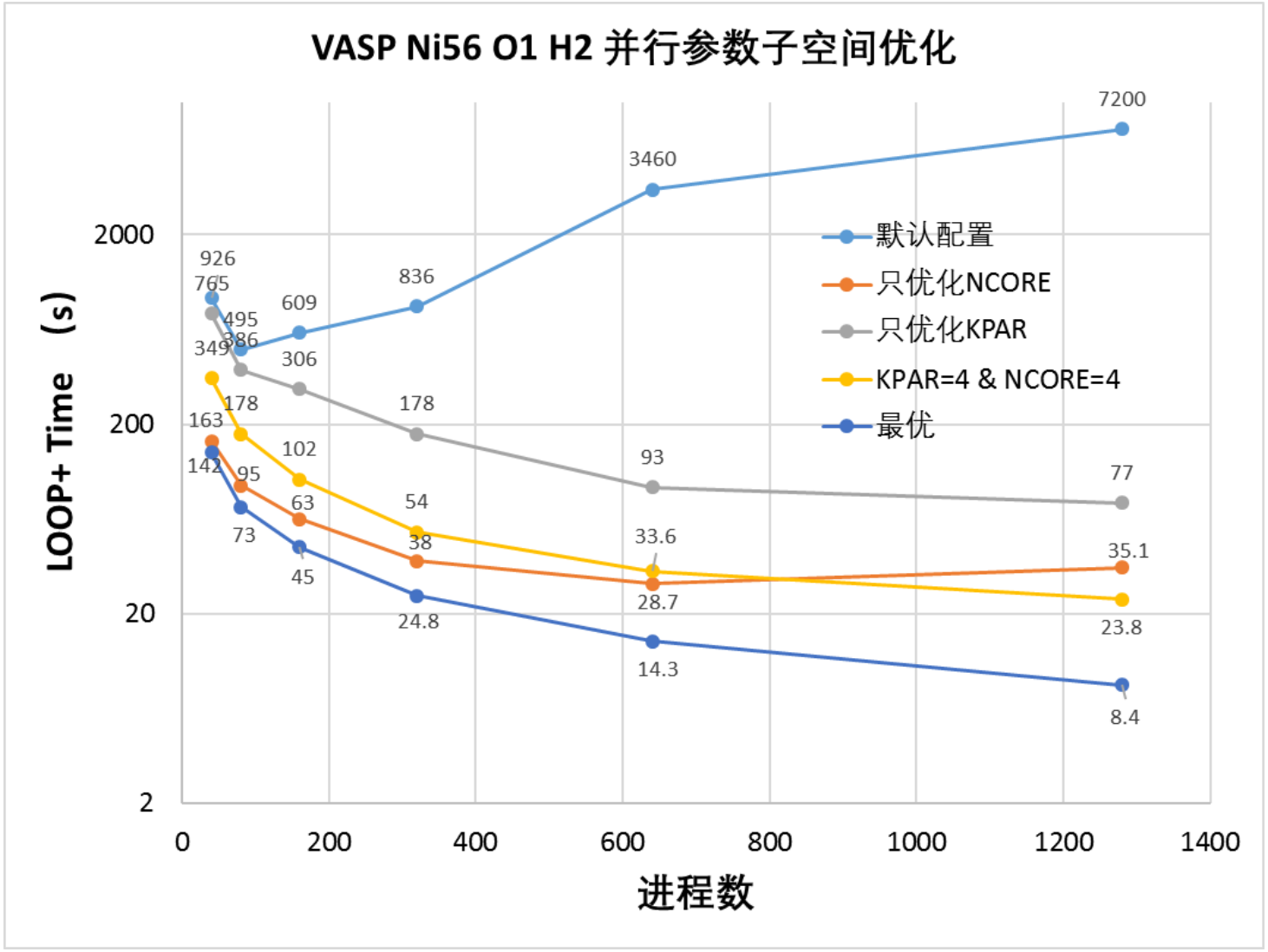
Geun Ho Gu

University of Delaware

- 计算之道
- 运行现状
- 优化之术

子空间遍历测试

	NCORE=	1	4	5	10	20	40
NP,40							
	KPAR1,	926.2,	407.5,	395.6,	214.1,	201.2,	163.5,
	KPAR2,	779.1,	349.2,	320.4,	160.6,	145.0,	nan,
	KPAR4,	765.3,	nan,	301.5,	142.4,	nan,	nan,
	KPAR8,	776.7,	nan,	302.5,	nan,	nan,	nan,
NP,80							
	KPAR1,	495.2,	213.8,	204.0,	112.7,	102.0,	95.2,
	KPAR2,	439.2,	189.8,	181.6,	87.8,	83.1,	82.3,
	KPAR4,	396.3,	178.9,	163.3,	78.2,	73.0,	nan,
	KPAR8,	386.4,	nan,	158.6,	72.8,	nan,	nan,
NP,160							
	KPAR1,	609.5,	127.9,	111.1,	63.5,	63.4,	63.1,
	KPAR2,	366.3,	105.9,	99.6,	60.4,	57.6,	57.4,
	KPAR4,	352.7,	101.8,	96.6,	55.4,	52.7,	48.7,
	KPAR8,	306.0,	92.0,	84.2,	48.9,	45.1,	nan,
NP,320							
	KPAR1,	836.8,	92.3,	76.8,	42.8,	37.8,	38.4,
	KPAR2,	298.7,	63.3,	55.3,	31.3,	32.8,	32.6,
	KPAR4,	185.2,	54.1,	50.9,	30.4,	28.7,	28.1,
	KPAR8,	178.7,	51.1,	48.6,	27.8,	26.2,	24.8,
NP,640							
	KPAR1,	3460.6,	167.3,	110.8,	43.2,	32.5,	28.7,
	KPAR2,	421.3,	45.6,	38.7,	21.2,	19.6,	19.4,
	KPAR4,	150.0,	33.6,	28.2,	16.0,	16.5,	15.9,
	KPAR8,	92.9,	27.8,	25.5,	15.0,	14.3,	14.3,
NP,1280							
	KPAR1,	nan,	575.9,	386.1,	100.5,	43.7,	35.1,
	KPAR2,	nan,	86.2,	59.7,	22.3,	16.7,	15.6,
	KPAR4,	209.0,	23.8,	19.9,	11.8,	10.2,	10.9,
	KPAR8,	76.5,	16.8,	14.9,	8.4,	8.8,	8.6,



In:MVION,ISTART=0,NELMDL=1,ALGO=,ISPIN=2,IMAGES=1; NodeCores=40,NNodes=40,
Out:bingam=0, NKPTS=8 ,NIONS=59 ,NBANDS=370 ,NGXYZ=40 56 126 ,ENCUT=520 ,LHFC=F ,LRPA=F ,IALGO=38

基于约化并行效率的自动化运行参数测试

优劣判定标准为，两两对比不同的作业参数配置，确定最优参数：

1. 当两个不同的作业参数配置的**运行成本相等时**：取计算时间最短（等于使用**通常的并行效率**）；
2. 当两个不同的作业参数配置的**运行成本不同时**：基于约化并行效率$\boldsymbol{E_r}$的值来判断。

约化并行效率$\boldsymbol{E_r}$：

$$E_r = {E_p}^{log^n_{R2/R1}} = (1/ECost)^{log^n_{ECost*SpeedUp}};$$

$$E_p = \frac{R1 \times T1}{R2 \times T2} = \frac{1}{ECost}$$

其中，case-1和case-2为同一应用作业对应的两个不同的作业参数配置，该应用作业采用case-1时的运行成本少于该应用作业采用case-2时的运行成本；R1为case-1对应的运行成本，R2为case-2对应的运行成本；T1为case-1对应的作业计算时长；T2为case-2对应的作业计算时长。

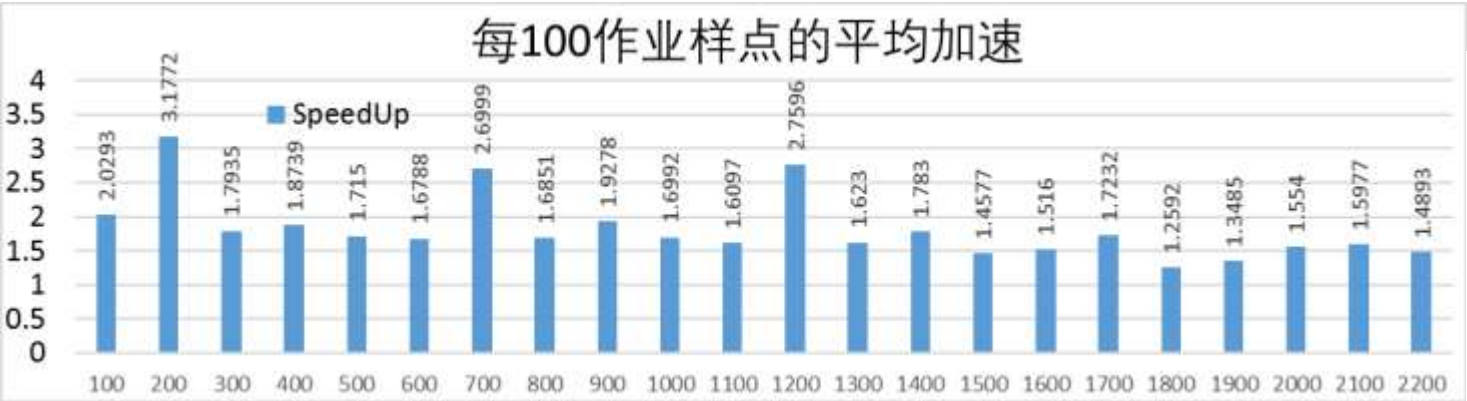
特别的，当n取2时，约化并行效率即为硬件成本每增大为2倍时的相应并行效率。

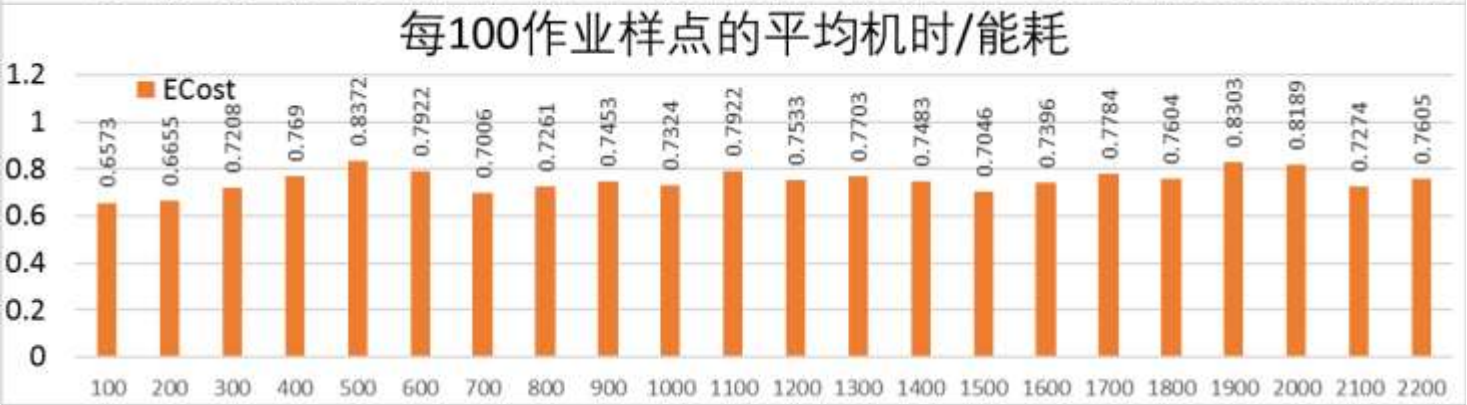
授权专利《一种应用于超算集群调度的作业运行参数优化方法》

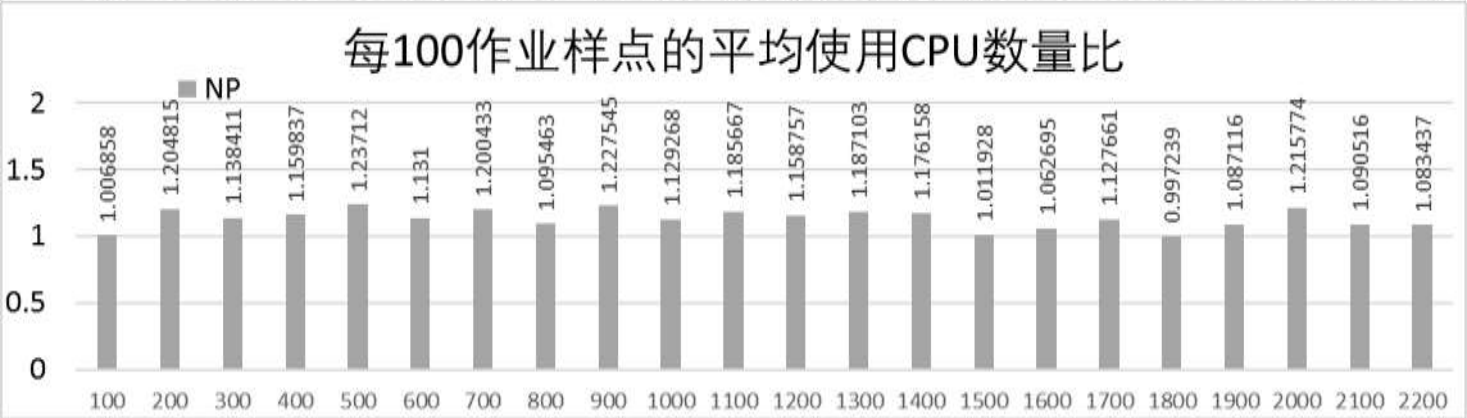
```
In:ALGO= ,ISPIN=2, Out:bingam=0,NKPTS=8    ,NIONS=59   ,NBANDS=384 ,NGXYZ=40  56  126 ,ENCUT=520 ,LHFC=F ,LRPA=F ,IALGO=38
Candidates: KPAR_list=(1 2 4 8), NPAR_list=(1 2 4 8 16 32 64 128 384), NCORE_list=(1 2 4 8 12 24 48), nini=0 1 2
SpeedUp=1.00 ,ECost=1.00 ,Tc=202   ,Tr=213   ,NP=96   ,KPAR=1   ,NPAR=96   ,NCORE=1   ,NewRaw,Original,OS20,NELOOP=2,NELMDL=1
SpeedUp=0.52 ,ECost=0.16 ,Tc=410   ,Tr=410   ,NP=8    ,KPAR=1   ,NPAR=2    ,NCORE=4   ,New,TstTest,OS20,x2EPLevels=(86 75 65)%
SpeedUp=0.97 ,ECost=0.17 ,Tc=219   ,Tr=219   ,NP=16   ,KPAR=2   ,NPAR=2    ,NCORE=4   ,New,NowBest,0,x2EP(KP1)=94%
SpeedUp=1.41 ,ECost=0.24 ,Tc=145   ,Tr=151   ,NP=32   ,KPAR=2   ,NPAR=4    ,NCORE=4   ,New,TmpBest,0,x2EP(NP1)=73%
SpeedUp=1.60 ,ECost=0.21 ,Tc=131   ,Tr=133   ,NP=32   ,KPAR=2   ,NPAR=2    ,NCORE=8   ,New,TmpBest,0,x2EP(NC1)=82%
SpeedUp=1.50 ,ECost=0.22 ,Tc=140   ,Tr=142   ,NP=32   ,KPAR=4   ,NPAR=2    ,NCORE=4   ,New,Discard,0,x2EP(KP1)=77%
SpeedUp=1.41 ,ECost=0.24 ,Tc=145   ,Tr=151   ,NP=32   ,KPAR=2   ,NPAR=4    ,NCORE=4   ,Old,Discard,0,x2EP(NP1)=73%
SpeedUp=1.60 ,ECost=0.21 ,Tc=131   ,Tr=133   ,NP=32   ,KPAR=2   ,NPAR=2    ,NCORE=8   ,Old,TmpBest,0,x2EP(NC1)=82%
SpeedUp=1.60 ,ECost=0.21 ,Tc=131   ,Tr=133   ,NP=32   ,KPAR=2   ,NPAR=2    ,NCORE=8   ,Old,SetBest,1,x2EP=82%
SpeedUp=3.04 ,ECost=0.22 ,Tc=69    ,Tr=70    ,NP=64   ,KPAR=4   ,NPAR=2    ,NCORE=8   ,New,NowBest,1,x2EP(KP1)=95% 并行效率73.2%
SpeedUp=4.53 ,ECost=0.22 ,Tc=47    ,Tr=47    ,NP=96   ,KPAR=4   ,NPAR=2    ,NCORE=12  ,New,NowBest,0,x2EP(NC1)=99% 并行效率72.6%
SpeedUp=4.53 ,ECost=0.22 ,Tc=47    ,Tr=47    ,NP=96   ,KPAR=4   ,NPAR=2    ,NCORE=12  ,BestP,OK,NPO=96,MVION

SpeedUp=1.00 ,ECost=1.00 ,Tc=182   ,Tr=184   ,NP=56   ,KPAR=1   ,NPAR=56   ,NCORE=1   ,NewRaw,Original,
SpeedUp=0.74 ,ECost=0.19 ,Tc=248   ,Tr=248   ,NP=8    ,KPAR=1   ,NPAR=2    ,NCORE=4   ,New,TstTest,
SpeedUp=1.32 ,ECost=0.22 ,Tc=139   ,Tr=139   ,NP=16   ,KPAR=1   ,NPAR=4    ,NCORE=4   ,New,NowBest,0,x2EP(NP1)=89%
SpeedUp=1.90 ,ECost=0.26 ,Tc=96    ,Tr=97    ,NP=28   ,KPAR=1   ,NPAR=4    ,NCORE=7   ,New,TmpBest,0,x2EP(NC1)=78%
SpeedUp=1.98 ,ECost=0.29 ,Tc=92    ,Tr=93    ,NP=32   ,KPAR=1   ,NPAR=8    ,NCORE=4   ,New,Discard,0,x2EP(NP1)=75%
SpeedUp=1.90 ,ECost=0.26 ,Tc=96    ,Tr=97    ,NP=28   ,KPAR=1   ,NPAR=4    ,NCORE=7   ,Old,TmpBest,0,x2EP(NC1)=78%
SpeedUp=1.90 ,ECost=0.26 ,Tc=96    ,Tr=97    ,NP=28   ,KPAR=1   ,NPAR=4    ,NCORE=7   ,Old,SetBest,1,x2EP=78%
SpeedUp=3.47 ,ECost=0.29 ,Tc=52    ,Tr=53    ,NP=56   ,KPAR=1   ,NPAR=8    ,NCORE=7   ,New,NowBest,1,x2EP(NP1)=92%
SpeedUp=6.57 ,ECost=0.30 ,Tc=27    ,Tr=28    ,NP=112  ,KPAR=1   ,NPAR=8    ,NCORE=14  ,New,NowBest,0,x2EP(NC1)=95%
```

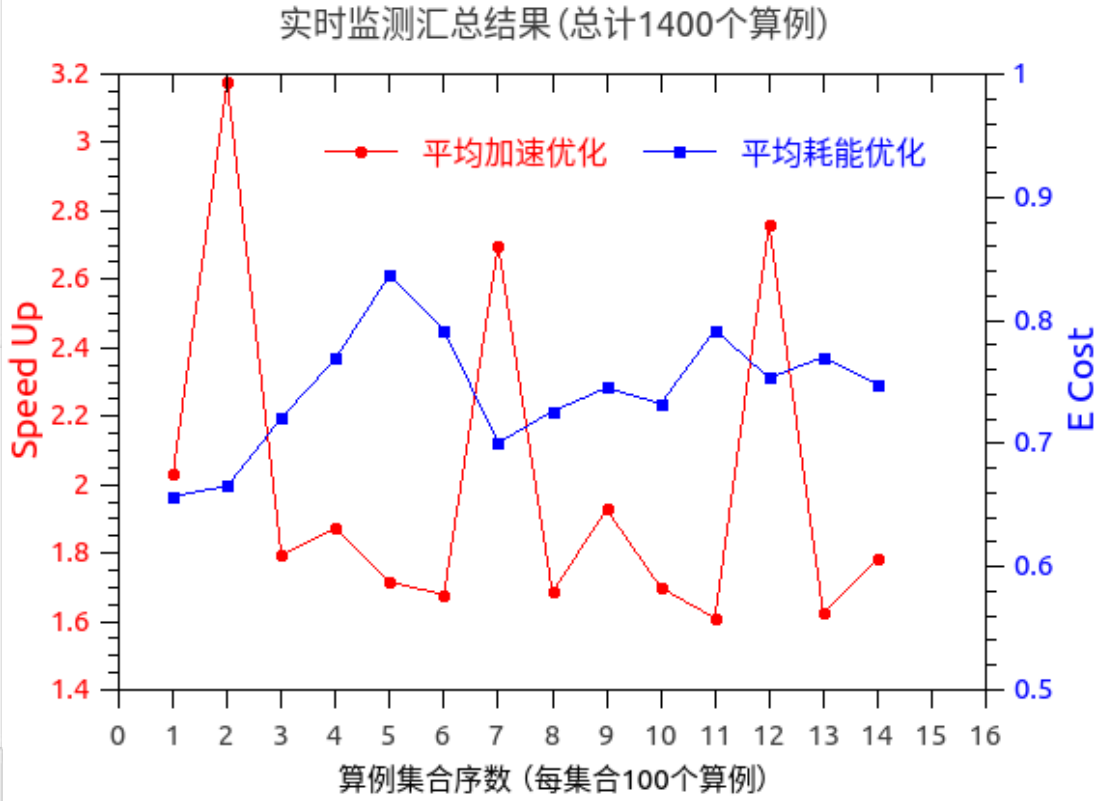
蓝色结果不很理想，其并行效率跟绿色结果相近，但绿色结果的明显更好：计算效率可以在某个临界点前后发生突变

VASP运行参数优化效果









每100个作业算例做统计平均，自动参数优化方法可使作业集平均加速约1.6 - 3.2倍，使作业计算机时平均减少15% - 35%。

VASP运行扩展性优化

表 1 Ni56O1H2 体系下的运行参数优化结果

Table1 Optimization results for Ni56O1H2 calculation

Processes Number	Parameter: KPAR*NPAR*NCORE	*Er*	Loop + Time (s)
80	1 * 80 * 1	-	514
160	2 * 8 * 10	-	63
320	2 * 8 * 20	88%	36
320	4 * 8 * 10	98%	32
640	4 * 16 * 10	94%	17
640	4 * 8 * 20	84%	19
640	8 * 8* 10	107%	15
1280	8 * 16 * 10	94%	8

VASP运行扩展性优化

表 2 Ni48C2O2 体系下的运行参数优化结果

Table2 Optimization results for Ni48C2O2 calculation

Processes Number	Parameter: KPAR*NPAR*NCORE	*Er*	Loop + Time (s)
200	1 * 4 * 10	-	906
200	4 * 2 * 5	-	515
400	4 * 2 * 10	143%	180
800	8 * 2 * 10	102%	88
800	4 * 4 * 10	85%	106
800	4 * 2 * 20	86%	105
1600	8 * 2 * 20	90%	49
1600	8 * 4 * 10	90%	49

VASP运行扩展性优化

表 3 C100I48Cr16 体系下的运行参数优化结果

Table2 Optimization results for C100I48Cr16calculatio

Processes Number	Parameter: KPAR*NPAR*NCORE	*Er*	Loop + Time (s)
80	1 * 80 * 1	-	1087
160	1 * 16 * 10	-	127
320	1 * 16 * 20	113%	56
1600	5 * 16 * 20	97%	12
800	1 * 40 * 20	73%	34
640	1 * 16 * 40	82%	34
1600	5 * 16 * 20	97%	12

面向 Web Portal 的运行时优化

当前：**提供运行参数自动化测试**

目标：**用户无需指定资源使用数量**。

代之以3个选项：1）效率优先 2）速度优先 3）均衡

面向 Slurm/LSF 命令行的运行时优化

```
#!/bin/bash

export PATH=/Path/To/OptimizationScripts:$PATH

module load vasp/6.3.2/openmpi_4.0.5_gcc10.2_hdf51.12.2

sbatch -p CPU -N 2 -n 80 opt_vasp vasp_std
```

适用多类算例

	GammaOnly?	NKPTS	NIONS	NBANDS	ENCUT	LHFC?	IALGO	NGX	NGY	NGZ
01	0	8	59	384	520	F	38	40	56	126
02	0	10	164	576	450	F	38	64	64	160
03	0	80	32	192	500	F	38	54	54	48
04	0	6	90	384	520	F	68	100	100	84
05	0	5	48	384	500	F	48	64	64	180
06	0	4	184	576	800	F	48	140	140	140
07	0	5	181	576	200	F	48	48	48	90
08	0	5	32	192	380	T	38	48	48	90
09	0	2	112	384	400	T	58	108	42	128
10	1	1	324	1260	400	F	38	84	70	80

AMD EPYC 7713 CPU Benchmark

```
In:MVION,ISTART=0,NELMDL=1,ALGO=VeryFast,ISPIN=1,IMAGES=1; NodeCores=128,NNodes=1,x2EPLevels=(95 88 60)
Out:bingam=0,NKPTS=4   ,NIONS=184 ,NBANDS=512 ,NGXYZ=140 140 140 ,ENCUT=800 ,LHFC=F ,LRPA=F ,IALGO=48 ,LPEAD=
Candidates: KPAR_list=(1 2 4), NPAR_list=(1 2 4 8 16 32 64 128 256 512), NCORE_list=(1 1 2 4 8 16 32 64 128 256), nini=(0 2 3
SpeedUp=1.00 ,ECost=1.00 ,Tc=175  ,Tr=175  ,NP=128 ,KPAR=1  ,NPAR=128 ,NCORE=1  ,OldRaw,Original,OS00,NELOOP=3,TimeCROK
SpeedUp=0.89 ,ECost=0.14 ,Tc=196  ,Tr=196  ,NP=16  ,KPAR=1  ,NPAR=4   ,NCORE=4  ,New,TestTest,OS00,NELOOP=3,TimeCROK
SpeedUp=1.51 ,ECost=0.17 ,Tc=116  ,Tr=116  ,NP=32  ,KPAR=1  ,NPAR=4   ,NCORE=8  ,New,TmpBest,NELOOP=3,ErL0,NC+1,Er=84%
SpeedUp=1.29 ,ECost=0.19 ,Tc=136  ,Tr=136  ,NP=32  ,KPAR=2  ,NPAR=4   ,NCORE=4  ,New,Discard,NELOOP=3,ErL0,KP+1,Er=72%
SpeedUp=1.29 ,ECost=0.19 ,Tc=136  ,Tr=136  ,NP=32  ,KPAR=1  ,NPAR=8   ,NCORE=4  ,New,Discard,NELOOP=3,ErL0,NP+1,Er=72%
SpeedUp=1.51 ,ECost=0.17 ,Tc=116  ,Tr=116  ,NP=32  ,KPAR=1  ,NPAR=4   ,NCORE=8  ,Old,SetBest,NELOOP=3,TimeCROK,2,x2EP=84%
SpeedUp=1.90 ,ECost=0.26 ,Tc=92   ,Tr=92   ,NP=64  ,KPAR=1  ,NPAR=4   ,NCORE=16 ,New,TmpBest,NELOOP=3,ErL2,NC+1,Er=63%
SpeedUp=1.55 ,ECost=0.32 ,Tc=113  ,Tr=113  ,NP=64  ,KPAR=2  ,NPAR=4   ,NCORE=8  ,New,Discard,NELOOP=3,ErL2,KP+1,Er=51%
SpeedUp=1.52 ,ECost=0.33 ,Tc=115  ,Tr=115  ,NP=64  ,KPAR=1  ,NPAR=8   ,NCORE=8  ,New,Discard,NELOOP=3,ErL2,NP+1,Er=50%
SpeedUp=1.90 ,ECost=0.26 ,Tc=92   ,Tr=92   ,NP=64  ,KPAR=1  ,NPAR=4   ,NCORE=16 ,Old,SetBest,NELOOP=3,TimeCROK,2,x2EP=63%
SpeedUp=1.88 ,ECost=0.53 ,Tc=93   ,Tr=93   ,NP=128 ,KPAR=1  ,NPAR=4   ,NCORE=32 ,New,TmpBest,NELOOP=3,ErL2,NC+1,Er=49%
SpeedUp=1.62 ,ECost=0.62 ,Tc=108  ,Tr=108  ,NP=128 ,KPAR=2  ,NPAR=4   ,NCORE=16 ,New,Discard,NELOOP=3,ErL2,KP+1,Er=43%
SpeedUp=1.61 ,ECost=0.62 ,Tc=109  ,Tr=109  ,NP=128 ,KPAR=1  ,NPAR=8   ,NCORE=16 ,New,Discard,NELOOP=3,ErL2,NP+1,Er=42%
SpeedUp=1.90 ,ECost=0.26 ,Tc=92   ,Tr=92   ,NP=64  ,KPAR=1  ,NPAR=4   ,NCORE=16 ,BestP,NELOOP=3,TimeCROK,NP0=128,MVION,User:r
```

双路单节点内，使用64核心相比使用128核心，速度更快

基于机器学习的VASP最佳运行参数预测

$ module show optkit/v1

/opt/Modules/tool/optkit/v1:

module-whatis

For VASP, use this tool to obtian "INCAR_best" with optimized KPAR/NPAR/NCORE for high efficient parallel and less crashes.

Usage:

1) load this module

2) do nothing inside the environment of SLURM job or export tjobid=$SLURM_JOB_ID or export nodeList="cnode01 cnode02 cnode03"

3) (optional) export mpiOtherArgs="...", to set MPI arguments except hostlist/hostfile

4) opt_vasp vasp_std

5) cp INCAR_best INCAR

6) mpirun vasp_std or mpirun $mpiOtherArgs --host "..." vasp_std

module load anaconda3_nompi

prepend-path PATH /opt/bin/optkit/

谢谢！

欢迎指正！

微信号：wszhang