信息处理学报 第0卷 1–7（??? 1992年）[DOI: 10.2197/ipsjjip.0.1]常规论文

超级计算机「富岳」上的并行排序性能评估

Tomoyuki Tokuue[1,a)]　Tomoaki Ishiyama[2,b)]

收稿日期：xx xx, xxxx，录用日期：xx xx, xxxx

摘要： 排序是最基本的算法之一。随着现代超级计算机每个节点的CPU核心数趋于增加，在高性能计算中，开发高度并行的排序程序正变得愈发重要。本研究实现了两种基于样本排序的多线程排序算法，并比较了它们在超级计算机「富岳」上的性能。第一种算法将输入序列划分为多个数据块，对每个数据块进行排序，然后从各数据块中按固定间隔抽样来选择枢轴。随后使用这些枢轴对每个数据块进行分区，并将不同数据块中的分区归并为一个有序序列。第二种算法与第一种算法的区别仅在于枢轴的选择：它使用二分搜索来选择枢轴，从而使每个分区中的元素数量相等。我们比较了这两种算法搭配不同顺序排序算法和多路归并算法时的性能。结果表明，在顺序排序中采用BlockQuicksort（一种通过减少条件分支来加速的快速排序），并在归并时采用选择树的第二种算法，对于各种输入数据类型和数据规模，都能持续实现较高的速度和并行效率。

关键词： 并行排序，超级计算机，多线程，性能评估

arXiv:2305.05245v1 [cs.DC] 9 May 2023

1. 引言

排序是最基础的算法之一，许多应用都需要快速排序程序。因此，共享/分布式内存环境中的排序优化与并行化始终是一个研究课题 [7, 11, 15, 16]。随着现代超级计算机中每个节点的计算核心数不断增加，开发高度并行的排序程序在高性能计算中变得愈发重要。例如，日本的旗舰超级计算机富岳每个节点配备了 48 个计算核心。尽管预计未来超级计算机的核心数还会进一步增加，但目前鲜有针对大量线程实现高并行效率的排序程序见诸报道。

本研究开发了可在富岳上高速运行并实现高效并行化的基于比较的排序程序。我们实现了两种基于样本排序的多线程排序算法 [3] ，并比较了各算法在富岳上处理不同输入数据类型和数据规模时的性能。我们还比较了样本排序中采用的顺序排序算法和多路归并算法，并找出最佳组合。对于顺序排序算法，我们采用快速排序 [4] 及其变体（例如 [2, 8]）。快速排序通常被认为比其他算法更快。对于多路归并，我们比较了使用和不使用数据结构的方法。我们采用二叉堆和选择树 [5] 作为数据结构，以实现高效归并。源代码已公开发布于 https://github.com/tmtoku/parallel_sortings。

本文结构如下：第 2 节介绍并行排序算法，并介绍本研究使用的顺序排序和多路归并算法。第 3 节评估并行排序算法在超级计算机「富岳」上的性能，第 4 节总结结果。

2. 并行样本排序

样本排序 [3], 广泛应用于并行环境，它通过采样选取多个枢轴，对输入序列进行分区，并对每个分区进行排序。

许多并行样本排序 [10, 12, 14] 针对一个包含 N 个元素的序列 A ，按以下四个步骤进行。
（1）对每个数据块进行排序 （2）选择枢轴 （3）分区 （4）对各分区进行多路归并

首先，它将 A 划分为 n_{B} 个块，并按顺序对每个块进行排序（步骤 1）。一个块是长度为 $\lceil N/n_{\mathrm{B}} \rceil$的连续子序列。然后，在步骤 2 中选择 $n_{\mathrm{P}} - 1$ 个枢轴P_1, ..., $P_{n_{\mathrm{P}}-1}$，并在步骤 3 中将每个块重新排列为 n_{P} 个分区。给定 $P_0 = -\infty$ 和 $P_{n_{\mathrm{P}}} = \infty$ （假设按升序排序），

[1] 千叶大学理学与工程研究生院数学与信息学部应用与认知信息学系[2] 千叶大学数字化转型推进委员会a)　t.tokuue@chiba-u.jpb)　ishiyama@chiba-u.jp

©1992年 日本信息处理学会

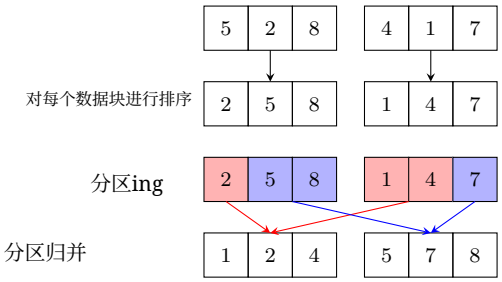


图 1：基于样本排序的并行排序示例。首先，它将输入序列划分为两个数据块，并对每个数据块进行排序。接下来，对每个已排序的数据块进行分区，在此示例中选择 4 作为枢轴。最后，收集每个数据块中的分区并将其合并。

第 k个分区（$0 \le k < n_{\mathrm{P}}$）是由 $P_k < x \le P_{k+1}$中元素 x 组成的连续子序列。最后，步骤 4 会归并所有块中的第k个分区，并将它们按顺序排列。步骤 1 和步骤 3 可以针对不同的块并行执行，步骤 2 则可以针对不同的枢轴并行执行。步骤 4 中每个分区的多路归并可以并行执行，但我们必须按顺序计算每个归并后分区相对于输出序列起始位置的偏移量。

图 1 展示了 $A = \{5, 2, 8, 4, 1, 7\}$、$n_{\mathrm{B}} = 2$和 $n_{\mathrm{P}} = 2$的情况示例。我们首先将 A 划分为两个各含三个元素的数据块，并分别对每个数据块排序（步骤 1）。假设选择 4 作为枢轴 P_1 （步骤 2），我们将每个数据块重新排列为两个分区：小于或等于 4 的分区和大于 4 的分区（步骤 3），然后合并这些分区并按顺序排列（步骤 4）。最终，输入序列被排序为$\{1, 2, 4, 5, 7, 8\}$。

由于第 1 步和第 4 步最耗时，优化这两部分对于加速样本排序至关重要。我们将快速排序的不同变体作为顺序排序进行比较，实现几种多路归并算法，并探索最快的组合。此外，样本排序的并行效率很大程度上取决于如何选择枢轴。我们还研究了不同枢轴选择方法对并行效率的影响，尤其是在许多元素重复时。

2.1 顺序排序

快速排序 [4] 是一种通过递归地重复进行分区来对整个序列进行排序的算法。分区是从输入序列 $A_0, \ldots, A_{N-1}$中选择一个枢轴元素 P，其中 N 表示元素数量，并重新排列元素，使得 $A_0, \ldots, A_k \le P$和 $A_k, \ldots, A_{N-1} \ge P$对于 $k \in \{0, \ldots, N-1\}$ 成立。快速排序的平均时间复杂度为 $O(N \log N)$。

假设每次分区时都选择输入序列中的最大或最小元素作为枢轴，那么快速排序的时间复杂度将达到最坏情况，并且为 $O(N^2)$。因此，内省排序 [6] 常用于将快速排序的最坏情况时间复杂度改善为 $O(N \log N)$。内省排序是一种算法，当快速排序的递归调用深度达到某个深度上限时，会从快速排序切换为堆排序，而该深度上限为 $O(\log N)$。如果深度上限为 $O(\log N)$，那么内省排序的最坏情况时间复杂度为 $O(N \log N)$。

为了利用处理器的指令流水线，即使程序包含条件分支，处理器也会尝试预测其中一个分支并推测执行。然而，现代处理器的流水线很长，预测不佳会降低程序性能。BlockQuicksort [2] 通过分块划分，避免分支预测错误导致快速排序性能下降。原始的划分方式 [4] 会从两端扫描输入序列，同时将元素与枢轴进行比较和交换。另一方面，分块划分会将待交换元素的位置存入缓冲区，并在扫描完成后进行交换。在现代 CPU 中，将元素与枢轴进行比较并存储其位置，可以在不使用条件分支的情况下完成。例如，ARMv8 处理器中的 CSET 或 CINC 指令可以实现这一操作，从而减少分区中的条件分支数量。

在模式败退快速排序 [8], 中，输入序列被划分为三部分：小于枢轴、等于枢轴和大于枢轴。当不同元素的数量 k 足够少时，排序时间为$O(Nk)$。大小严重偏斜的分区会降低性能，这种情况称为不良划分。当不良划分的数量达到 $\lfloor \log N \rfloor$时，模式败退快速排序会停止递归调用，从而比内省排序更及时地切换到堆排序。

2.2 归并

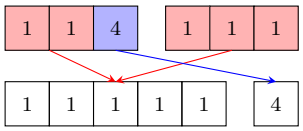
我们可以使用数据结构高效地归并 n_{B} 已排序的序列。将每个已排序序列的首元素组织在二叉堆或选择树中 [5] ，并按升序取出元素，即可得到一个已排序序列。由于取出最小元素并更新数据结构的计算成本为 $O(\log n_{\mathrm{B}})$，因此归并所需的总时间为 $O(n_{\mathrm{M}} \log n_{\mathrm{B}})$，其中 n_{M} 是要在一个分区中归并的元素总数。

顺序排序无需使用数据结构即可实现多路归并。我们可以将 n_{B} 个已排序序列复制到输出数组中并对其排序，从而将它们合并为一个已排序序列。尽管这种方法的时间复杂度为$O(n_{\mathrm{M}} \log n_{\mathrm{M}})$，但由于内存访问局部性，其缓存效率应该更高。

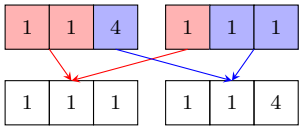
2.3 枢轴选择与分区

除非选择枢轴，使各线程待归并元素的总数均衡分配，否则样本排序的并行效率会降低。规则采样并行排序 (PSRS) 从每个数据块中采样 [10] 个元素，将其排序，并从 n_{B}（$n_{\mathrm{P}} - 1$）个样本中选取 $n_{\mathrm{P}} - 1$ 个枢轴。它以固定间隔从已排序序列中选取样本和枢轴，使待归并元素的总数尽可能均衡。枢轴选择的时间复杂度为 O（$n_{\mathrm{B}} n_{\mathrm{P}}$ log（$n_{\mathrm{B}} n_{\mathrm{P}}$）），对应于顺序

©1992年日本信息处理学会



(a) 使用枢轴 $P_1 = 1$ 进行分区。由于第一个分区中的元素多于第二个分区，因此各分区之间的归并时间严重失衡。



(b) 在 PSES 中使用 $P_1 = 1$ 和 $c_1 = 3$进行分区。第一个分区包含小于 1 的元素和三个 1；第二个分区包含剩余的两个 1 和大于 1 的元素。由于两个分区的元素数量相同，因此各分区之间的归并时间均衡。

图 2：不同元素较少时的分区。

对所有样本进行排序。

SORTLIB [*1]，这是一种适用于 A64FX 和其他架构、速度相当快的样本排序实现。它从每个数据块中随机抽取元素，而不对数据块进行排序，对所有样本进行排序，并以固定间隔选取枢轴。由于样本总数与元素数量成正比 N，与线程数 t 成反比，因此枢轴选择的时间复杂度为 $O((N/t)\log(N/t))$。

当大量元素的值重复时，仅使用枢轴进行分区可能会导致各分区之间的元素总数严重失衡。例如，对于输入序列$A = \{1,1,4,1,1,1\}$，无论选择哪些枢轴，在 PSRS 中，一个分区所包含的元素总数最大值都大于或等于 5（**图 2a**）。假设各分区中的元素总数不平衡，则每次归并所需的时间也会严重失衡，并且随着线程数增加，并行效率会降低。并行精确划分排序（PSES）[12]可以解决这一问题。该方法选择满足公式（1）的枢轴 P_k，并计算公式（2）中定义的数量 c_k。

$$|\{x \in A \mid x < P_k\}| \leq k\frac{N}{n_{\mathrm{P}}} \leq |\{x \in A \mid x \leq P_k\}| \quad (1)$$

$$c_k = k\frac{N}{n_{\mathrm{P}}} - |\{x \in A \mid x < P_k\}| \quad (2)$$

第 k 个分区（$0 \leq k < n_{\mathrm{P}}$）包含所有满足 x sat- isfying $P_k < x < P_{k+1}$ 的元素，以及等于 P_{k+1}的 c_k 个元素。无论不同元素的数量有多么少，该方法都能使每个分区中的元素总数相等。通过使用二分搜索，统计每个数据块中小于 P_k 和等于 P_k 的元素数量后，检查是否满足公式 (1)。因此，可以在 $O(n_{\mathrm{B}} \log N \log\lceil N/n_{\mathrm{B}}\rceil)$ 时间内确定 P_k 和 c_k。在 $A = \{1,1,4,1,1,1\}$ 和 $n_{\mathrm{P}} = 2$ 的示例中，小于 1 的元素有 0 个，而小于或等于 1 的元素有 5 个。

1. 将 P_1 设置为 1、将 c_1 设置为 3 后，如 **图 2b** 所示，每个分区中的元素总数相等，从而使各分区的合并开销均衡。

3. 性能评估

我们通过实验表明，在超级计算机「富岳」上，我们的并行排序程序显著优于一种常用程序。

3.1 实验设置

我们使用理化学研究所计算科学中心（R-CCS）的超级计算机「富岳」评估了并行排序实现的性能。富岳的一个节点配备一个A64FX处理器和32 GB内存。该处理器由四个核心内存组（CMG）组成，运行频率为2.00 GHz。每个CMG拥有12个计算核心，因此A64FX总共有48个计算核心。

我们实现了 PSRS 和 PSES 作为并行排序算法。数据块 n_{B} 和分区 n_{P} 的数量与性能评估中使用的线程数相同。对于包含许多重复元素的输入数组，PSES 提高了多路归并的并行效率，但其选择枢轴的成本高于 PSRS。因此，我们定量比较了在重复元素较多和较少的输入下两者的性能差异。我们还将选择树实现为多路归并的数据结构，并将其与二叉堆进行比较；二叉堆是在 [12]中更常用的数据结构。

该程序使用 C++ 编写，并采用 OpenMP 线程并行化；使用富士通编译器 FCC 4.9.0（clang模式）和 -Ofast优化选项进行编译。我们使用 numactl命令为每个线程分配一个计算核心。这里，我们将前 12 个线程绑定到 CMG0 中的计算核心，将接下来的 12 个线程绑定到 CMG1 中的计算核心。同样，我们将剩余的 24 个线程分配到 CMG2 和 CMG3。输入和输出数组被划分为与线程数相同数量的数据块，并且每个数据块都分配在距离拥有该数据块的线程最近的 CMG 中。

我们针对 **表1** 中所示的六种输入序列，以及元素数量的不同取值 N（$N = 10^7$ 和 10^8），评估了排序实现的性能。我们通过交换从零到 $N-1$ 的递增序列中随机选取的$\sqrt{N}$ 个元素的位置，构造了近乎有序数据。由于 Duplicate3 的元素只能取三个值，因此其中包含许多重复元素。Particle 是一个结构体序列，每个结构体包含用于排序的键，以及表示三维空间中粒子的数据（质量、位置、速度、加速度和势能；双精度下共 88 位），这些都是引力 N 体模拟中粒子结构体的典型组成部分。我们使用这种数据类型来评估按某个键对粒子进行排序的性能。我们对每种排序测量了 20 次运行时间，并以其平均值作为实现的性能。

[*1] https://github.com/jmakino/sortlib

©1992年 日本信息处理学会

表1: 输入序列中一个元素的数据类型和数据大小。我们根据键（无符号64位整数 t）对对和粒子进行排序。

序列	Type	Size (Byte)	
均匀整数	无符号 32 位整数 t	4	$[0, 2^{32}-1]$ 中均匀随机分布的 32 位整数。
均匀浮点数	浮点数	4	$[0, 1$ 中均匀随机分布的 32 位浮点数）。
近乎有序	无符号 32 位整数 t	4	由 $[0, N-1]$ 个 32 位整数组成的近乎有序序列。
Duplicate3	无符号 32 位整数 t	4	$\{0, 1, 2\}$ 中的均匀随机整数。
Pair	结构体	16	键索引对。这些键是 $[0, 2^{64}-1]$ 中的 64 位均匀随机整数。
粒子	结构体	96	键和粒子数据。键是 $[0, 2^{64}-1]$ 中的 64 位均匀随机整数。

3.2 并行排序算法的比较

图 3 展示了并行排序算法的运行时间。我们在 PSRS 和 PSES 中都使用 BlockQuicksort 进行顺序排序，并使用我们的选择树进行多路归并。这里，gnu parallel::sort 是 GNU C++ 标准库-libstdc++ 中并行模式下 [13] 的一种并行多路归并排序 [14]实现。它的排序过程与 PSES 类似：使用 std::sort 进行顺序排序，并在 k大于四时使用选择树进行 k路归并。不过，当存在许多重复元素时，它不一定会使每个分区中的元素数量相等。这是为了使排序稳定而对枢轴的位置施加额外限制所致。

对于具有 $N = 10^7$ 的粒子，使用 gnu parallel::sort 排序所需的时间不会随着线程数超过 24 而减少。相比之下，PSRS 和 PSES 的排序时间会随着线程数增加而单调下降，直至 48 个线程。对于所有具有 $N = 10^7$ 和 10^8 的输入序列，使用 48 个线程时，PSES 的速度是 gnu parallel::sort 的两倍以上。对于重复元素较少的输入序列，PSRS 的排序时间与 PSES 几乎相同。然而，对于 Duplicate3，无论元素数量是多少，线程数超过 4 后，PSRS 的运行时间都不会再减少，这与 PSES 形成鲜明对比。这种饱和现象是因为当线程数大于不同元素的数量时，无论如何选择枢轴，每个线程需要合并的元素数量都会严重失衡，如第 2.3 节所述。

图 **4** 进一步凸显了这些趋势，展示了并行效率。对于所有输入序列，PSES 在线程数超过 12 时都具有相对出色的并行效率；当线程数增至 48 时，使用 $N = 10^8$ 和 10^7，分别仍保持在约 0.5 和 0.4。除 Duplicate3 外，PSRS 的并行效率几乎与 PSES 相同；但在 Duplicate3 上使用 48 个线程时，其效率最低，约为 PSES 的十分之一。对于除 Duplicate3 以外的输入序列，gnu parallel::sort 的并行效率最低。在近乎有序输入上，其并行效率尤其低，这表明针对近乎有序输入，单线程顺序排序比其他输入更快。

当线程数较多时，所有算法在 $N = 10^8$ 下的并行效率都与 $N = 10^7$ 相近或更高。本研究对 N 中时间复杂度相对较高的部分进行了并行化，即对每个数据块进行排序和分区合并。因此，在增加 N 而保持线程数不变的情况下，时间复杂度在并行部分中会相对更加突出而非并行部分中，从而使较大的 N 具有更高的并行效率。尤其是对于 Pair 和 Particle，由于其数据量大于其他对象，随着 N 增加，并行效率会提高，因为并行部分中的数据复制耗时更长。

3.3 顺序排序算法比较

图 5 展示了 PSES 使用不同算法对各数据块进行顺序排序时的运行时间。我们比较了三种算法：内省排序、模式败退快速排序和 BlockQuicksort；在所有情况下，多路归并分区时均使用我们的选择树。先前的研究 [12] 认为，由于快速排序的执行时间会随输入序列而变化，因此归并排序更适合对数据块进行排序。然而，上述三种算法都改善了快速排序的最坏情况时间复杂度，而且通常比归并排序快得多，因此本研究对它们进行了比较。我们使用 libstdc++ 中的 std::sort 作为内省排序的实现，并使用 Boost 库中的 boost::sort::pdqsort 作为模式败退快速排序的实现。BlockQuicksort 的公开代码 *2 提供了多种实现，例如采用不同的枢轴选择方式；我们使用其中一种代码：blocked double pivot check mosqrt::sort。

对于除 Duplicate3 之外的输入序列，BlockQuicksort 的速度最快。对于 Duplicate3，模式败退快速排序的速度最快，因为正如第 2.1 节所述，它专门针对不同元素数量较少的情况进行了优化，能够在这种情况下实现更快的运行速度。在大多数情况下，std::sort 的运行速度并不比其他算法快。

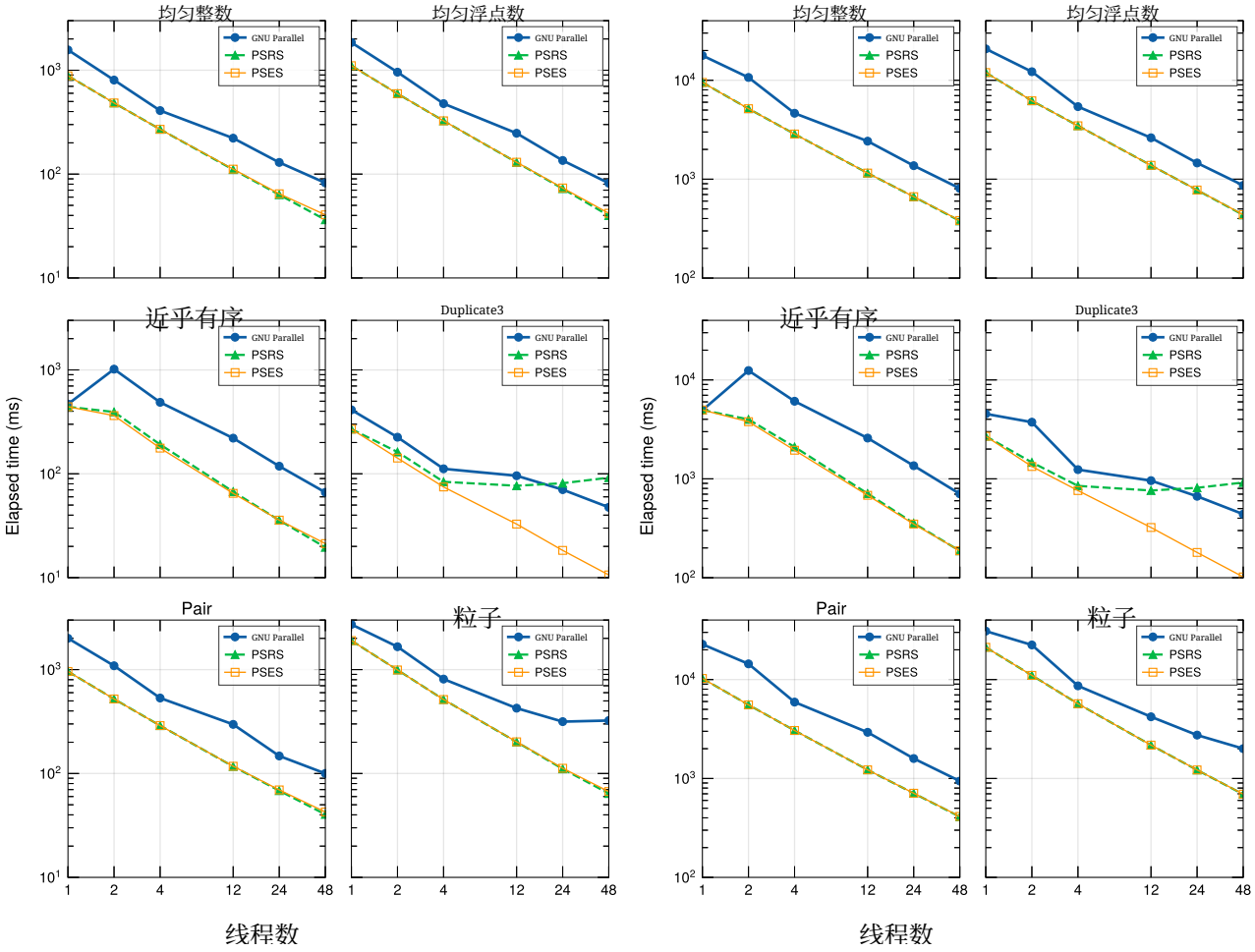
3.4 多路归并算法的比较

图 6 展示了 PSES 使用不同多路归并算法归并分区所需的运行时间。我们比较了三种算法：二叉堆、选择树，以及不使用数据结构的顺序排序。我们使用 libstdc++ 中的 std::priority queue 实现二叉堆，并使用 std::sort 顺序排序来归并分区。在所有情况下，我们都使用 std::sort 对每个数据块进行排序。

我们的选择树实现对于所有输入序列来说都是最快的。不过，当线程数达到 24 或更多时，对于 UniformInt、UniformFloat 和 Duplicate3 输入序列，std::sort 的性能与选择树相近，尽管 std::sort 通常比选择树更频繁地移动元素。其性能相对较好的原因可能是，该排序算法

*2 https://github.com/weissan/BlockQuicksort

©1992年 日本信息处理学会



(a) $N = 10^7$。 (b) $N = 10^8$.

图 3：并行排序算法的运行时间。我们比较了 PSRS 和 PSES 的实现，以及 libstdc++ 中的 gnu parallel::sort。在 PSRS 和 PSES 中，我们均使用 BlockQuicksort 进行顺序排序，并使用我们的选择树进行多路归并。

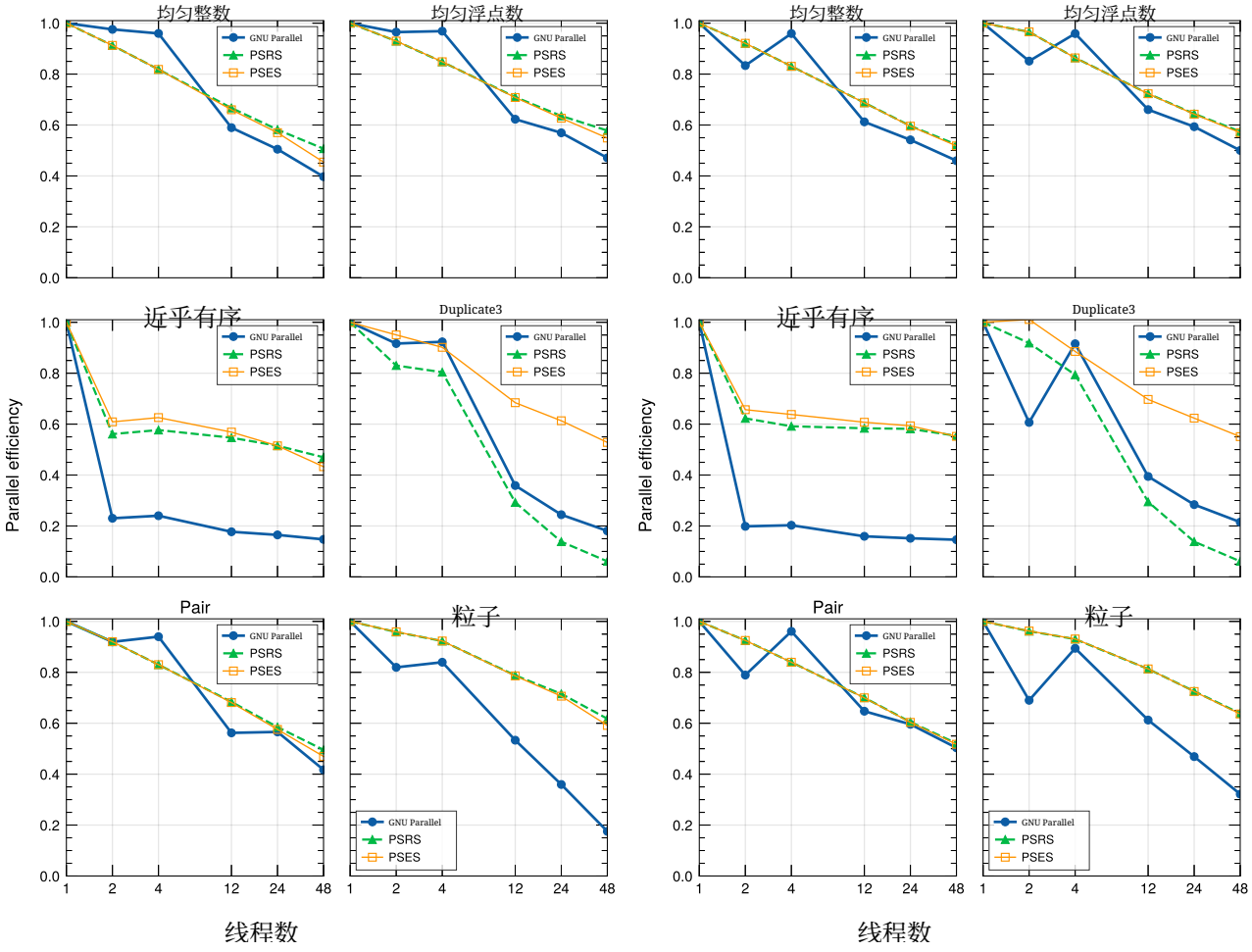
不使用外部数据结构，而且缓存高效。另一方面，对于 Pair 和 Particle，该排序算法的速度会变慢，因为这些数据的体量较大，导致缓存性能不佳。线程数达到 12 或更多时，对于近乎有序输入，std::sort 的速度比 UniformInt 慢 1.5 倍以上，尽管选择树在这两种输入上的表现相近。我们使用 Fujitsu 提供的性能分析器，检查了 std::sort 对这两种输入进行归并时的差异，发现 std::sort 在处理近乎有序输入时，执行时间显著增加。此外，对于近乎有序输入，std::sort 最终调用了堆排序，而对于 UniformInt 则没有。因此，我们认为，近乎有序输入中待归并元素的顺序可能会降低 std::sort 的性能。对于 Duplicate3、Pair 和 Particle，std::priority queue 的排序时间在线程数从 2 增至 4 时反而上升。在大多数情况下，使用 std::priority queue 进行多路归并比使用 std::sort 更慢，这表明 libstdc++ 中的容器并未针对在富岳上高效运行而实现。

4. 结论

我们实现了两种多线程排序算法，可在富岳等现代超级计算机上执行。这两种算法均基于样本排序，但有所不同。在枢轴的选择方式上存在差异。第一种算法将输入序列划分为多个数据块，然后分别对每个数据块进行排序，并通过按固定间隔从各数据块中采样来选择枢轴。第二种算法则使用二分搜索选择枢轴，以确保每个分区中的元素数量相等。随后，我们比较了这两种算法与不同顺序排序和多路归并算法相结合时的性能。采用 BlockQuicksort 进行顺序排序，并使用选择树进行多路归并时，第二种算法在各种输入数据类型和数据规模下均表现出较高的运行速度和并行效率。

在未来的工作中，我们将考虑对并行基数排序和原地并行排序算法进行性能评估。基数排序可能比比较排序更快。目前，代表性的原地并行排序算法包括并行快速排序和原地并行超标量样本排序（IPS^{4}o） [1]。IPS^{4}o 是一种新颖的并行排序算法，通过递归执行多路分区，改进了超标量样本排序的方法 [9]。它是一种原地算法，因为每个递归层级的额外内存用量与输入序列的长度无关。本研究实现的两个并行排序程序均假设输入序列和输出序列

©1992年 日本信息处理学会



(a) $N = 10^7$。 (b) $N = 10^8$。

图 4：并行排序算法的并行效率。我们比较了 PSRS 和 PSES 的实现，以及 libstdc++ 中的 gnu parallel::sort。对于 PSRS 和 PSES，我们均使用 BlockQuicksort 进行顺序排序，并使用我们的选择树进行多路归并。

在内存中分别存放。换言之，这些程序需要为输出序列分配与输入序列大小相同的额外内存。对于每个节点内存较少的超级计算机（如富岳），或字节浮点比（B/F）较低、因而高效利用缓存至关重要的系统，开发高性能的原地并行排序程序十分重要。

致谢本研究得到了日本千叶大学 IAAR研究支持计划、文部科学省/日本学术振兴会科学研究费助成事业（项目编号 JP21H01122）、文部科学省“超级计算机「富岳」研究推进计划”（JP-MXP1020200109）以及 JICFuS 的支持。

参考文献

[1] Axtmann, M.、Witt, S.、Ferizovic, D. 和 Sanders, P.：《原地（共享内存）排序算法的工程实现》，计算研究资料库（CoRR）（2020年9月）。[2] Edelkamp, S. 和 Weiß, A.：《BlockQuicksort：避免快速排序中的分支预测错误》，*mics* 24 2019 *ACM J. Exp. Algorith.*，第（）卷。[3] Frazer, W. D. 和 McKellar, A. C.：《样本排序：一种用于最小存储树排序的采样方法》，*J. ACM*，第17卷，第3期，第496–507页（在线），DOI：10.1145/321592.321600（1970年）。

[4] Hoare, C. A. R.：快速排序，*The Computer Journal*，第 5 卷，第 1 期，第 10–16 页（1962年）。[5] Knuth, D. E.：*The Art of Computer Programming Volume 3: (2nd Ed.) Sorting and Searching*，Addison Wesley Longman Publishing Co., Inc.，美国（1998年）。[6] MUSSER, D. R.：内省排序与选择算法，*Software: Practice and Experience*，第 27 卷，第 8 期，第 983–993 页（1997年）。[7] Obeya, O.、Kahssay, E.、Fan, E. 和 Shun, J.：理论高效且实用的并行原地基数排序，*The 31st ACM Symposium on Parallelism in Algorithms and Architectures*，SPAA '19，美国纽约州纽约市，计算机协会，第 213–224 页（在线），DOI：10.1145/3323165.3323198（2019年）。[8] Peters, O. R. L.：模式败退快速排序，*CoRR*，第 abs/2106.05123 卷（2021年）。[9] Sanders, P. 和 Winkel, S.：超标量样本排序，*Algorithms – ESA 2004*（Albers, S. 和 Radzik, T.，编），柏林、海德堡，Springer Berlin Heidelberg，第 784–796 页（2004年）。[10] Shi, H. 和 Schaeffer, J.：规则采样并行排序，*Journal of Parallel and Distributed Computing*，第 14 卷，第 4 期，第 361–372 页（1992年）。[11] Shun, J.、Blelloch, G. E.、Fineman, J. T.、Gibbons, P. B.、Kyrola, A.、Simhadri, H. V. 和 Tangwongsan, K.：简要公告：基于问题的基准测试套件，*Proceedings of the Twenty-Fourth Annual ACM Symposium on Parallelism in Algorithms and Architectures*，SPAA '12，美国纽约州纽约市，计算机协会，第 68–70 页（在线），DOI：10.1145/2312005.2312018（2012年）。[12] Siebert, C. 和 Wolf, F. G. E.：使用精确划分的可扩展并行排序算法，技术报告，亚琛（2011年）。

©1992年日本信息处理学会

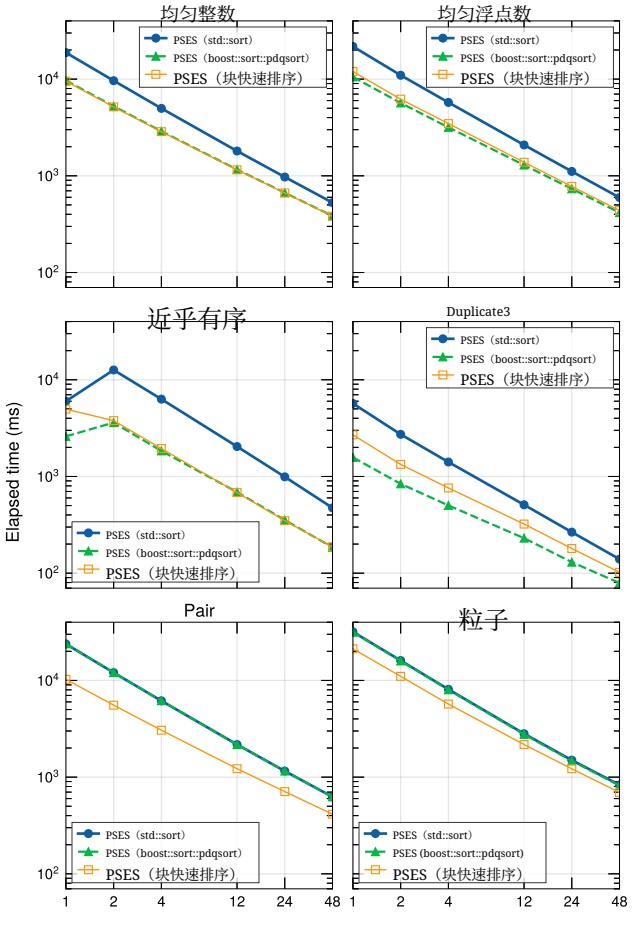


图 5：使用不同块排序算法时 PSES ($N = 10^8$) 的运行时间。我们比较了 libstdc++ 中的 std::sort、Boost 库中的 boost::sort::pdqsort，以及 BlockQuicksort。多路归并采用我们实现的选择树。

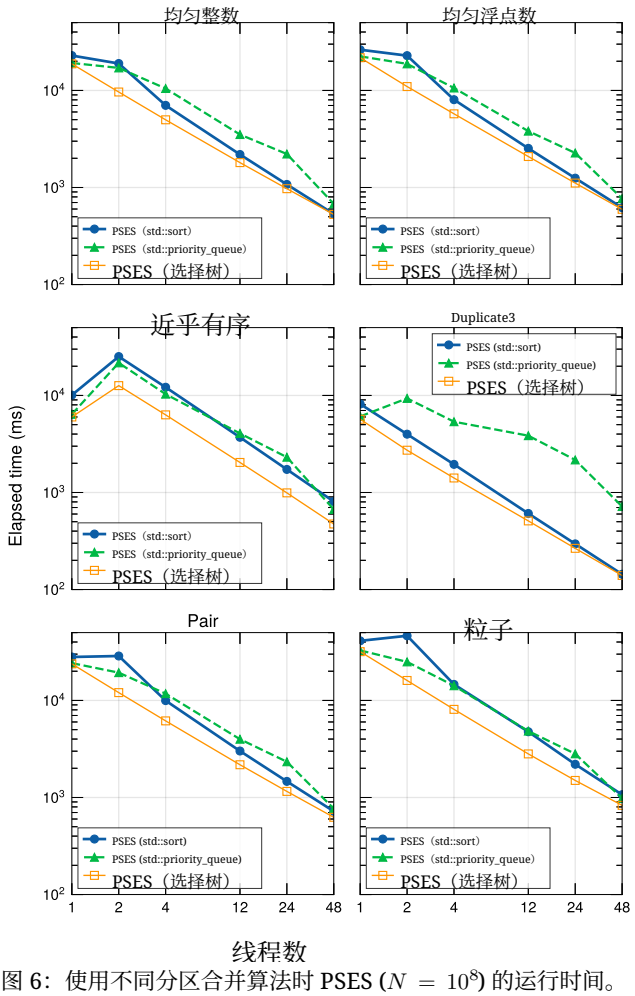


图 6：使用不同分区合并算法时 PSES ($N = 10^8$) 的运行时间。我们比较了 libstdc++ 中的 std::priority queue 和我们实现的选择树，以及不使用数据结构、通过 std::sort 进行的顺序排序。每个数据块均使用 std::sort 排序。

[13] Singler, J. 和 Konsik, B.：GNU Libstdc++ 并行模式：软件工程考量，*Proceedings of the 1st International Workshop on Multicore Software Engineering*，IWMSE '08，美国纽约州纽约，计算机协会，第 15–22 页（2008 年）。[14] Singler, J.、Sanders, P. 和 Putze, F.：MCSTL：多核标准模板库，*Proceedings of the 13th International Euro-Par Conference on Parallel Processing*，Euro-Par'07，柏林，海德堡，施普林格出版社，第 682–694 页（2007 年）。[15] Solomonik, E. 和 Kalé, L. V.：高度可扩展的并行排序，*2010 IEEE International Symposium on Parallel and Distributed Processing (IPDPS)* 1 12 ，第 – 页（在线），DOI：10.1109/IPDPS.2010.5470406（2010 年）。[16] Tsigas, P. 和 Zhang, Y.：快速排序的简单快速并行实现及其在 SUN Enterprise 10000 *Eleventh Euromicro Conference on Parallel Distributed and Network-Based Processing 2003. Proceedings.* 372 381 2003 上的性能评估，，第 – 页（）。

©1992年 日本信息处理学会