Задание 0 – Описание вычислительного узела

```
ProLiant XL270d Gen10
PRETTY_NAME="Ubuntu 22.04.5 LTS"
NAME="Ubuntu"
VERSION_ID="22.04"
VERSION="22.04.5 LTS (Jammy Jellyfish)"
VERSION_CODENAME=jammy
ID=ubuntu
ID_LIKE=debian
HOME_URL="https://www.ubuntu.com/"
SUPPORT_URL="https://help.ubuntu.com/"
BUG_REPORT_URL="https://bugs.launchpad.net/ubuntu/"
PRIVACY_POLICY_URL="https://www.ubuntu.com/legal/terms-and-policies/privacy-policy"
UBUNTU_CODENAME=jammy
available: 2 nodes (0-1)
node 0 cpus: 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 40 41 42 43 44 45 46
47 48 49 50 51 52 53 54 55 56 57 58 59
node 0 size: 385636 MB
node 0 free: 268775 MB
node 1 cpus: 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 60 61 62
63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79
node 1 size: 387008 MB
node 1 free: 193443 MB
node distances:
node   0   1
  0:  10  21
  1:  21  10
Architecture:             x86_64
  CPU op-mode(s):         32-bit, 64-bit
  Address sizes:          46 bits physical, 48 bits virtual
  Byte Order:             Little Endian
CPU(s):                   80
  On-line CPU(s) list:    0-79
Vendor ID:                GenuineIntel
  Model name:             Intel(R) Xeon(R) Gold 6248 CPU @ 2.50GHz
    CPU family:           6
    Model:                85
    Thread(s) per core:   2
    Core(s) per socket:   20
    Socket(s):            2
    Stepping:             7
    CPU max MHz:          3900.0000
    CPU min MHz:          1000.0000
    BogoMIPS:             5000.00
    Flags:                fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge m
                          ca cmov pat pse36 clflush dts acpi mmx fxsr sse sse2 s
                          s ht tm pbe syscall nx pdpe1gb rdtscp lm constant_tsc
                          art arch_perfmon pebs bts rep_good nopl xtopology nons
                          top_tsc cpuid aperfmperf pni pclmulqdq dtes64 monitor
                          ds_cpl smx est tm2 ssse3 sdbg fma cx16 xtpr pdcm pcid
                          dca sse4_1 sse4_2 x2apic movbe popcnt tsc_deadline_tim
                          er aes xsave avx f16c rdrand lahf_lm abm 3dnowprefetch
                           cpuid_fault epb cat_l3 cdp_l3 invpcid_single intel_pp
                          in ssbd mba ibrs ibpb stibp ibrs_enhanced fsgsbase tsc
                          _adjust bmi1 avx2 smep bmi2 erms invpcid cqm mpx rdt_a
                           avx512f avx512dq rdseed adx smap clflushopt clwb inte
                          l_pt avx512cd avx512bw avx512vl xsaveopt xsavec xgetbv
                          1 xsaves cqm_llc cqm_occup_llc cqm_mbm_total cqm_mbm_l
                          ocal dtherm ida arat pln pts hwp hwp_act_window hwp_pk
                          g_req pku ospke avx512_vnni md_clear flush_l1d arch_ca
                          pabilities
```

Задание 1 – Умножения матрицы на вектор с параллельной инициализацией массивов

M = N	Количество потоков															
	1		2		4		7		8		16		20		40	
	T1	**S1**	**T2**	**S2**	**T4**	**S4**	**T7**	**S7**	**T8**	**S8**	**T16**	**S16**	**T20**	**S20**	**T40**	**S40**
20000 (3052 MiB)	*0,61*	*1,04*	*0,32*	*1,95*	*0,16*	*3,88*	*0,09*	*6,81*	*0,08*	*7,78*	*0,04*	*15,20*	*0,03*	*18,99*	*0,02*	*33,66*
40000 (12207 MiB)	*2,49*	*0,91*	*1,25*	*1,80*	*0,73*	*3,09*	*0,41*	*5,75*	*0,37*	*6,09*	*0,17*	*13,76*	*0,15*	*15,95*	*0,07*	*31,43*

Анализируя коэффициент ускорения Sp(n) для различных размеров матриц и числа потоков, можно сделать следующие выводы о масштабируемости программы:

1. Рост ускорения при увеличении числа потоков

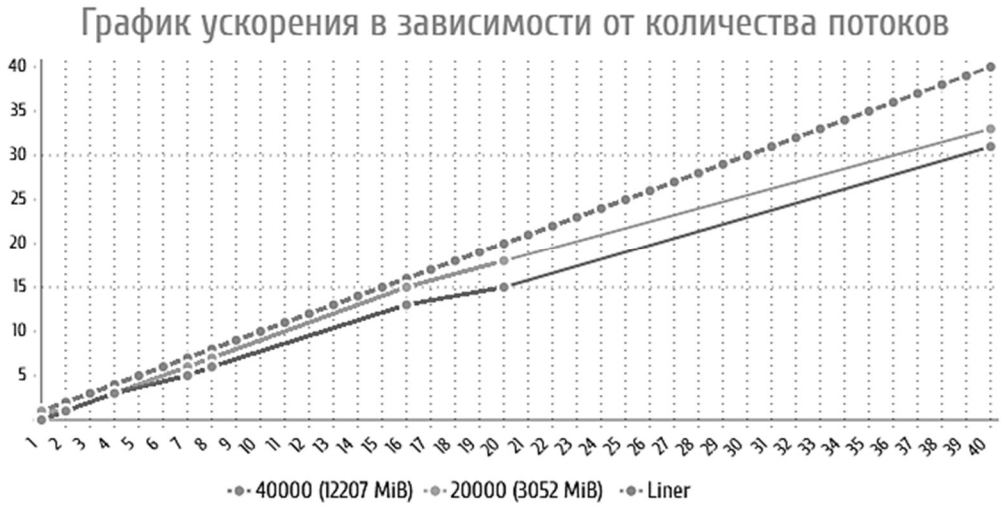
Ускорение растет с увеличением количества потоков, но не линейно. Это типично для параллельных программ, так как накладные расходы на управление потоками и доступ к памяти могут снижать эффективность.

2. Уменьшение эффективности при большом числе потоков

При 40 потоках ускорение (например, Sp(40)=33.66 для 20000×20000 и Sp(40)=31.43 для 40000×40000) значительно меньше, чем теоретически ожидаемые 40 раз. Это указывает на накладные расходы и возможные узкие места в системе (например, ограничение пропускной способности памяти).

3. Нелинейность ускорения

Для малых чисел потоков (1, 2, 4) ускорение близко к линейному, но при дальнейшем росте начинает снижаться. Например, при 8 потоках Sp(8)=7.78 для 20000×20000, что довольно близко к идеальному Sp(8)=8. Однако при 16 потоках ускорение уже составляет Sp(16)=15.20, что указывает на возрастающие накладные расходы.



Задание 2 – Параллельная версия программы численного интегрирования

Анализируя коэффициент ускорения Sp(n) для различного числа потоков, можно сделать следующие выводы о масштабируемости программы:

1. Линейное ускорение наблюдается до 16 потоков, после чего рост эффективности замедляется.
2. При увеличении количества потоков более 16 эффект от распараллеливания падает: прирост ускорения снижается, хотя время выполнения все еще уменьшается. Это может быть связано с:
    a. накладными расходами OpenMP на управление потоками;
    b. ограничением пропускной способности памяти (memory bandwidth);
    c. несбалансированной загрузкой процессора.
3. При 40 потоках ускорение всего 10.63 вместо ожидаемых 40, что говорит об ограничении параллелизма в данной задаче.

Задание 3 – Параллельная реализация решения системы линейных алгебраических уравнений

Сравнение вариантов параллельной реализации, оба варианта показывают значительное ускорение при увеличении числа потоков, но:

1. Вариант 1 (отдельные #pragma omp parallel for для каждого цикла) работает быстрее при увеличении числа потоков. Например, при 80 потоках он достигает времени 6.12 секунд (при 65 потоках) – 10.12 секунд (при 79 потоках).
2. Вариант 2 (единая #pragma omp parallel секция) показывает несколько худшие результаты. При 80 потоках лучшее время – 7.21 секунд (при 68 потоках) – 13.47 секунд (при 80 потоках).

Таким образом, Вариант 1 оказывается более эффективным для данной задачи.

Анализ эффективности schedule(...)

1. schedule(static) даёт лучшие результаты, особенно при небольших chunk_size:
    a. Лучший результат: schedule(static, 8) – 8.98 секунд.
    b. В целом, schedule(static) дает меньшую изменчивость времени выполнения.
2. schedule(dynamic) работает хуже:
    a. Лучший результат 15.69 секунд (schedule(dynamic, 256)), что хуже, чем у static.

Выводы

1. Вариант 1 (#pragma omp parallel for) предпочтительнее, так как он даёт лучшие результаты.
2. Использование schedule(static, 8) или schedule(static, 64) даёт наибольшее ускорение.
3. Число потоков: Оптимальное количество потоков находится в диапазоне 50–70. После этого производительность начинает деградировать.

Общий вывод по заданиям:

Проведенный анализ масштабируемости параллельных алгоритмов показывает, что увеличение количества потоков приводит к значительному ускорению вычислений, однако эффективность распараллеливания зависит от ряда факторов, таких как накладные расходы на управление потоками, пропускная способность памяти и балансировка нагрузки.