Pertemuan 16 — UAS: Tugas Besar

“Satu soal, makin banyak prosesor” — Capaian B: *High Performance Computing*

Hendri Karisma

Semester Ganjil 2026/2027

Program Studi Teknik Informatika, STMIK Tazkia

Alur pertemuan hari ini

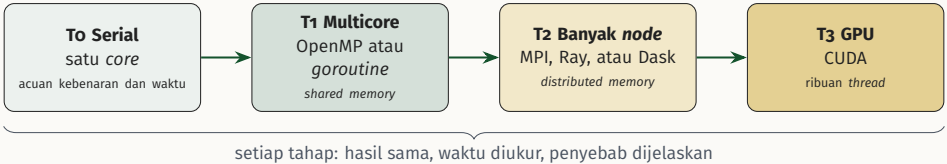
1. Tujuan dan kaitan dengan Capaian B
2. Pilihan soal dan ukuran *input*
3. Empat tahap wajib
4. Verifikasi kebenaran
5. Protokol pengukuran
6. Grafik yang diharapkan
7. Analisis penyebab
8. Nilai lebih: keandalan, *cost*, energi
9. Repositori, laporan, rubrik, aturan
10. Pertanyaan yang akan muncul saat demo

Bobot dan cakupan

UAS bernilai **45%** nilai akhir, mengukur CPMK-4, CPMK-5, dan CPMK-6, dan mencakup materi pertemuan 9–15: *performance metric*, CPU dan OpenMP, MPI, GPU dan CUDA, paralelisme data besar, dan *AI cluster*.

Tujuan

Satu soal, empat cara memecahnya



Ringkasan

Sasaran: *pecahkan satu soal komputasi berat dengan makin banyak prosesor, ukur berapa cepat yang benar-benar didapat, lalu jelaskan mengapa tidak secepat yang dibayangkan.*

Kaitan dengan CPMK

CPMK	Kemampuan yang diukur	Bukti di tugas besar
CPMK-4	*Performance metric*, CPU dan memori, OpenMP, MPI	Tahap T0–T2, kurva *speedup*, efisiensi, *strong*/*weak scaling*, analisis *cache* dan komunikasi
CPMK-5	GPU, CUDA, *heterogeneous computing*	Tahap T3, *kernel* naif dan teroptimasi, waktu transfer terpisah, *roofline*
CPMK-6	Pola paralel tingkat tinggi (*task*, *actor*), keandalan, *cost*, energi	Pilihan Ray atau Dask untuk T2, opsi *checkpoint*, estimasi *cost* dan energi

Berbeda dengan UTS yang menilai arsitektur layanan, UAS menilai **seberapa baik Anda memahami ke mana waktu komputasi pergi**.

Pilihan soal

Empat pilihan soal

Soal	Kompleksitas	Saran ukuran *input*	Ciri yang menarik
Perkalian matriks $C = AB$	$O(N^3)$ operasi, $O(N^2)$ data	$N = 1024$ s.d. 8192, FP32 ($N = 4096$: ≈ 137 GFLOP)	*Compute-bound*; *tiling* dan *cache* sangat berpengaruh; ada pembanding cuBLAS
Konvolusi citra (filter)	$O(WHk^2)$	Citra 4096^2 s.d. 16384^2 piksel, *kernel* $k = 3$ s.d. 15	*Memory-bound* untuk k kecil; *halo* saat dibagi ke banyak *node*
Simulasi *n-body*	$O(N^2)$ per langkah	$N = 10.000$ s.d. 100.000 benda, 10–100 langkah	Komunikasi semua-ke-semua; cocok untuk melihat *cost* MPI
Aggregation data besar	$O(n)$ baca, *shuffle* saat *group by*	1–10 GB data publik (misalnya catatan perjalanan taksi atau data cuaca)	*I/O-bound*; T2 dengan Dask atau Ray; T3 dengan RAPIDS atau *kernel reduction*

$O(\cdot)$: orde pertumbuhan jumlah operasi atau data terhadap ukuran soal (N, n, $W \times H$; k: ukuran *filter*).

Soal lain boleh diajukan sampai minggu ke-11 bila setara beratnya dan dapat dikerjakan pada keempat tahap. Contoh: *k-means*, *stencil* panas 2D, Monte Carlo.

Memilih ukuran *input* yang tepat

- **Terlalu kecil** — To selesai dalam milidetik; *overhead* membuat *thread*, mengirim *message* MPI, atau menyalin ke GPU lebih besar daripada kerjanya. Kurva *speedup* terlihat jelek dan tidak bermakna.
- **Terlalu besar** — To butuh berjam-jam atau tidak muat di memori laptop, dan eksperimen tidak bisa diulang.
- ***Baseline*** — pilih ukuran sehingga To berjalan **30 detik sampai 5 menit**. Lalu siapkan 3–4 ukuran untuk *weak scaling*.
- Hitung kebutuhan memori dulu. Contoh: tiga matriks FP32 berukuran 8192^2 butuh $3 \times 8192^2 \times 4$ byte $\approx$ 805 MB.

Ringkasan

Ukuran *input* adalah bagian dari desain eksperimen. Tuliskan alasan memilihnya di laporan.

Empat tahap wajib

Tahap T0 dan T1: satu mesin

T0 — Serial

- Ditulis dalam C, C++, Rust, atau Go. Python murni hanya sebagai pembanding tambahan, bukan acuan.
- Kompilasi dengan *optimization* (`-O2` atau `-O3`); catat *flag*-nya.
- Hasilnya menjadi acuan kebenaran dan acuan T_1 untuk semua *speedup*.

T1 — *Multicore*

- OpenMP (C/C++) atau *goroutine* dengan *worker pool* (Go).
- Ukur dengan 1, 2, 4, ..., jumlah *core* fisik.
- Minimal satu perbaikan sadar-*cache*: urutan *loop*, *blocking*, atau tata letak data, dengan pengukuran sebelum dan sesudahnya.
- Waspadai *false sharing* dan *hyper-threading*.

Laptop dengan 4 *core* sudah cukup. Catat model CPU, jumlah *core* fisik dan logis, serta ukuran *cache*-nya (`lscpu`).

Tahap T2 dan T3: banyak *node* dan GPU

T2 — Banyak *node*

- **MPI** (Open MPI atau MPICH) untuk matriks, konvolusi, *n-body*; atau **Ray**/**Dask** (Python) untuk *aggregation* data dan *task farm*.
- Boleh memakai beberapa *container* di satu laptop, beberapa laptop anggota kelompok dalam satu jaringan, atau VM.
- Ukur dengan 1, 2, 4 proses/*worker*; pisahkan waktu komputasi dan waktu komunikasi.
- Jujur: *container* di satu laptop berbagi *core* yang sama. Jelaskan dampaknya.

T3 — GPU

- CUDA C/C++ di Google Colab atau Kaggle (GPU gratis); Numba/CuPy hanya bila *kernel*-nya ditulis sendiri.
- Wajib dua versi: *kernel* **naif** dan *kernel* **teroptimasi** (*shared memory*, *tiling*, *coalescing*).
- Ukur dengan CUDA *event*; pisahkan waktu salin *host*→*device*, *kernel*, dan *device*→*host*.
- Bandingkan dengan *library* (cuBLAS, cuDNN, atau RAPIDS) bila ada.

Verifikasi kebenaran: cepat tetapi salah = nol

- Setiap versi (T1, T2, T3) dibandingkan dengan *output* T0 pada *input* yang sama.
- Untuk bilangan pecahan, hasil tidak akan identik bit per bit karena urutan penjumlahan berbeda. Gunakan *relative error*:

$$\frac{\|C_{\text{paralel}} - C_{\text{serial}}\|}{\|C_{\text{serial}}\|} < \varepsilon, \qquad \varepsilon \approx 10^{-5} \text{ (FP32, soal kecil)}$$

  (ε dibaca *epsilon*: batas toleransi; $\|\cdot\|$: norma, ukuran besar suatu matriks.) *Error* FP32 tumbuh bersama ukuran soal dan jumlah langkah: untuk matriks besar atau *n-body* panjang, 10^{-4}–10^{-3} wajar. Tulis alasan pilihan ε.
- Untuk *aggregation* data: jumlah baris, jumlah total, dan beberapa nilai kelompok harus sama persis.
- Sediakan satu skrip `verify.sh` yang menjalankan semua versi pada *input* kecil dan melaporkan LULUS atau GAGAL.

Ringkasan

Kebenaran diperiksa **sebelum** kecepatan. Versi yang gagal verifikasi tidak dihitung di grafik apa pun.

Protokol pengukuran

Metric yang wajib dilaporkan

- ***Speedup*** $S(p) = T_1/T_p$ (T_1: waktu To serial, T_p: waktu dengan p prosesor)
- **Efisiensi** $E(p) = S(p)/p$
- ***Strong scaling*** — ukuran *input* tetap, p bertambah.
- ***Weak scaling*** — beban per prosesor tetap, ukuran *input* tumbuh bersama p. Idealnya waktu tetap.
- **GFLOPS** atau **GB/s** yang dicapai, dibandingkan dengan puncak teoretis *hardware*.

Estimasi fraksi serial (Karp–Flatt)

Dari *speedup* terukur S pada p prosesor:

$$e = \frac{1/S - 1/p}{1 - 1/p}$$

Bila e tetap ketika p naik, penghambatnya bagian serial (Amdahl). Bila e ikut naik, penghambatnya *overhead* yang tumbuh: komunikasi, *synchronization*, atau beban tidak seimbang.

Cara mengukur yang jujur

1. **Catat lingkungan** — model CPU/GPU, jumlah *core*, RAM, versi *compiler*, *flag*, versi CUDA, jenis GPU Colab yang didapat.
2. ***Warm-up*** — satu kali jalan dibuang (*initialization* CUDA, *cache*, JIT pada JVM, *startup* Ray/Dask).
3. **Ulangi** — minimal **5 kali** per titik; laporkan median, min, dan maks.
4. **Ukur bagian yang sama** — tentukan apakah waktu baca *file* dan *initialization* masuk hitungan, lalu konsisten di semua versi.
5. **Pisahkan komponen waktu** — komputasi, komunikasi (MPI/*shuffle*), dan transfer GPU dicatat terpisah.
6. **Simpan *raw data*** — satu CSV per eksperimen di `results/`, plus skrip yang menghasilkannya dan skrip yang menggambar grafiknya.

Ringkasan

GPU Colab berganti-ganti jenisnya. Jalankan semua pengukuran T3 dalam satu sesi, dan tuliskan jenis GPU yang dipakai.

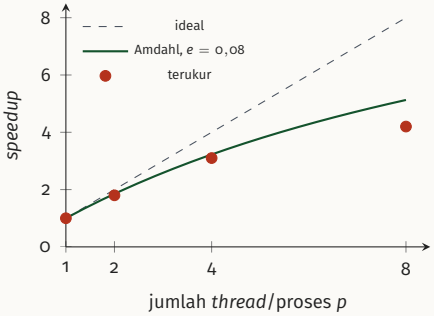
Bentuk tabel ringkasan

Tahap	*Configuration*	Waktu (s)	*Speedup*	Efisiensi	GFLOPS	Lulus verifikasi
T0	serial, -O3	…	1,00	1,00	…	acuan
T1	OpenMP, 4 *thread*	…	…	…	…	ya/tidak
T1	OpenMP + *blocking*, 4 *thread*	…	…	…	…	ya/tidak
T2	MPI, 4 proses	…	…	…	…	ya/tidak
T3	CUDA naif (tanpa / dengan transfer)	…	…	–	…	ya/tidak
T3	CUDA *tiling* (tanpa / dengan transfer)	…	…	–	…	ya/tidak
T3	cuBLAS (pembanding)	…	…	–	…	ya/tidak

Efisiensi GPU tidak dihitung per *thread*, karena “jumlah prosesor” GPU tidak sebanding dengan *core* CPU. Untuk GPU, bandingkan GFLOPS atau GB/s terhadap puncak teoretis kartunya.

Grafik yang diharapkan

Grafik 1 dan 2: *strong scaling*



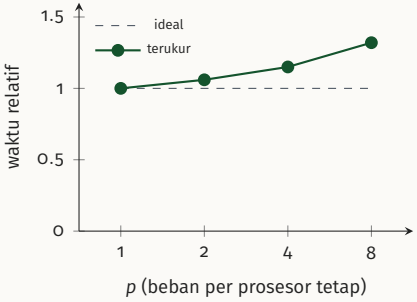
Ilustrasi, bukan hasil nyata.

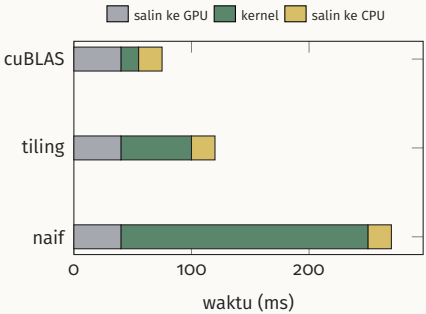
Grafik yang diminta:

1. *Speedup* terhadap p untuk T1 dan T2, dengan garis ideal dan kurva Amdahl dari fraksi serial hasil estimasi Anda.
2. Efisiensi terhadap p.

Pada contoh ini titik $p = 8$ jatuh di bawah kurva Amdahl. Itu tanda ada penghambat lain selain bagian serial, misalnya *bandwidth* memori yang sudah jenuh, atau 8 *thread* berjalan di 4 *core* fisik. Penjelasan seperti inilah yang dinilai.

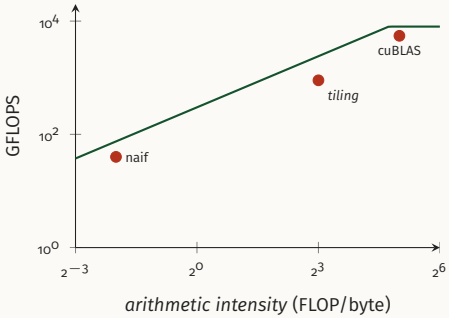
Grafik 3 dan 4: *weak scaling* dan rincian waktu GPU





Ilustrasi, bukan hasil nyata. Perhatikan bahwa pada *kernel* yang sudah cepat, transfer data mendominasi.

Grafik 5 (nilai lebih): *roofline*



Ilustrasi, bukan hasil nyata.

- Garis miring: dibatasi *bandwidth* memori. Garis datar: dibatasi puncak komputasi.
- *Arithmetic intensity* = jumlah FLOP dibagi jumlah byte yang dibaca dari memori utama.
- *Tiling* menaikkan *intensity* karena data dipakai ulang dari *shared memory*; titiknya bergeser ke kanan dan ke atas.
- Ambil puncak dan *bandwidth* dari lembar spesifikasi GPU/CPU yang Anda pakai.

Analisis penyebab

Angka saja tidak cukup: jelaskan mengapa

Gejala	Kemungkinan penyebab	Cara membuktikan
Speedup mendatar di T1	*Bandwidth* memori jenuh; bagian serial; *false sharing*	Hitung GB/s yang dicapai; *profiler* (`perf`); ubah tata letak data
Versi *blocking* jauh lebih cepat	*Cache miss* berkurang	`perf stat -e cache-misses` sebelum dan sesudah
T2 lebih lambat dari T1	Komunikasi lebih mahal daripada kerja per *node*; *container* berbagi *core*	Rincian waktu komunikasi vs komputasi; model α–β (*alpha–beta*)
GPU kalah dari CPU	*Input* kecil; transfer PCIe mendominasi	Grafik rincian waktu; ulangi dengan *input* lebih besar
Kernel naif lambat	Akses memori tidak *coalesced*; data tidak dipakai ulang	Nsight Systems dan Nsight Compute; *roofline*

Ringkasan

Setiap klaim penyebab harus didukung satu pengukuran tambahan. “Mungkin karena *cache*” tanpa bukti hanya mendapat sebagian nilai.

Nilai lebih: keandalan, *cost*, dan energi

Keandalan (*checkpoint*)

Untuk *n-body* atau komputasi panjang: simpan *state* tiap k langkah, matikan satu proses di tengah jalan, lanjutkan dari *checkpoint* terakhir. Ukur *overhead checkpoint* dan waktu yang terselamatkan. Kaitkan dengan studi kasus *training* Llama 3 di pertemuan 15.

Cost dan energi

Perkirakan *cost* menjalankan versi terbaik di *cloud* (harga per jam instans CPU vs GPU) untuk ukuran *input* 10 kali lebih besar. Perkirakan energi dari daya rancangan (TDP) atau `nvidia-smi`. Mana yang paling murah per hasil, bukan per jam?

Ringkasan

Di dunia nyata, pertanyaan yang lebih sering muncul adalah “mana yang cukup cepat dengan *cost* dan energi paling kecil”.

Repositori, laporan, dan penilaian

Struktur repositori dan laporan

```
uas-kelompok-XX/
  README.md       # lingkungan, cara jalan
  Makefile
  data/           # generator masukan
  src/
    t0_serial/  t1_openmp/
    t2_mpi/     t3_cuda/
  verify.sh
  bench/
    run_all.sh
    results/      # CSV mentah
    plot.py
  notebooks/      # Colab untuk T3
  docs/laporan.pdf
  AI-USAGE.md
```

Laporan maksimal 12 halaman:

1. Soal, alasan memilih, dan ukuran *input*
2. Lingkungan *hardware* dan *software*
3. Desain tiap tahap: cara membagi kerja dan data
4. Verifikasi kebenaran
5. Metodologi pengukuran
6. Hasil: tabel ringkasan dan grafik 1–4 (5 opsional)
7. Analisis penyebab, didukung pengukuran tambahan
8. Nilai lebih (bila ada)
9. Kesimpulan: tahap mana paling layak untuk soal ini, dan mengapa
10. Lampiran: pembagian kerja, catatan penggunaan AI

Presentasi dan demo

Format per kelompok: 20 menit

Presentasi hasil dan analisis	8 menit
Demo langsung	5 menit
Tanya jawab	7 menit

Demo wajib memperlihatkan

1. `verify.sh` berjalan dan semua versi LULUS
2. T0 dan T1 berjalan langsung, dengan waktu tercetak
3. T3 berjalan di Colab/Kaggle
4. Satu grafik dibuat ulang dari data mentah

Ringkasan

Setiap anggota wajib menjelaskan minimal satu tahap. Nilai individu dapat berbeda dari nilai kelompok bila kontribusi tidak seimbang.

Rubrik penilaian

Komponen	Bobot	Nilai penuh bila
Implementasi T0–T3	25%	Keempat tahap berjalan; T1 punya perbaikan sadar-*cache*; T3 punya versi naif dan teroptimasi
Verifikasi kebenaran	10%	`verify.sh` ada dan semua versi lulus dengan toleransi yang dijelaskan
Pengukuran	20%	Protokol jujur, 5 kali ulang, *strong* dan *weak scaling*, transfer GPU terpisah, data mentah
Analisis penyebab	20%	Setiap gejala dijelaskan dan dibuktikan dengan pengukuran tambahan; Karp–Flatt dipakai
Laporan	10%	Struktur lengkap, grafik terbaca, kesimpulan menjawab pertanyaan
Presentasi dan demo	15%	Demo berhasil; semua anggota menjawab dengan paham
Nilai lebih	+10%	*Roofline, checkpoint dan pemulihan, estimasi cost dan energi, pembanding library*

Nilai akhir UAS dibatasi 100. UAS bernilai 45% dari nilai mata kuliah.

Aturan main dan jadwal

Aturan

- Kelompok 3–4 orang; boleh sama dengan kelompok UTS.
- Alat bantu AI boleh dipakai; tuliskan di `AI-USAGE.md`. Setiap anggota harus mampu menjelaskan *kernel* dan kode paralelnya.
- Kode dari sumber lain (termasuk contoh CUDA dari buku PMPP) wajib disebut sumbernya dan dibedakan dari kode sendiri.
- Memalsukan atau menyalin angka pengukuran: nilai UAS nol.
- Terlambat: dikurangi 10 poin per hari, maksimal 3 hari.

Jadwal

Minggu 10	Tugas diumumkan; pilih soal
Minggu 11	T0 dan T1 jalan; ukuran *input* ditetapkan
Minggu 12	T2 jalan; `verify.sh`
Minggu 13	T3 naif; rencana tugas besar dinilai
Minggu 14	T3 teroptimasi; semua pengukuran
Minggu 15	Presentasi progres
Minggu 16	**Presentasi akhir dan demo**
Minggu 16 + 3 hari	*Deadline* repositori dan laporan

Checklist pengumpulan

- ☐ Tautan repositori dapat diakses dosen
- ☐ `README.md`: lingkungan, cara kompilasi, cara jalan
- ☐ Empat tahap T0–T3 ada di `src/`
- ☐ `verify.sh` dan hasilnya
- ☐ *Notebook* Colab/Kaggle untuk T3
- ☐ CSV mentah di `bench/results/`
- ☐ Grafik 1–4 dan skrip pembuatnya
- ☐ Tabel ringkasan dengan GFLOPS atau GB/s
- ☐ Estimasi fraksi serial (Karp–Flatt)
- ☐ Laporan PDF $\leq$ 12 halaman
- ☐ `AI-USAGE.md` dan pembagian kerja
- ☐ *Slide* presentasi

Pertanyaan yang akan muncul saat demo

1. Berapa fraksi serial program Anda, dan bagian kode mana itu?
2. Mengapa *speedup* dengan 8 *thread* tidak 8 kali?
3. T2 Anda berjalan di satu laptop. Apakah itu benar-benar terdistribusi? Apa yang akan berubah di *cluster* sungguhan dengan InfiniBand?
4. Berapa persen waktu GPU habis untuk transfer data? Kapan GPU tidak layak dipakai?
5. Tunjukkan di *kernel* Anda bagian yang memakai *shared memory*. Mengapa itu mempercepat?
6. Di mana posisi *kernel* Anda pada *roofline*? Dibatasi komputasi atau memori?
7. Bila Anda punya anggaran 1 juta rupiah per bulan, versi mana yang dijalankan, dan di mana?
8. Tunjukkan satu baris kode yang Anda tulis sendiri dan jelaskan.

Menutup semester

Paruh pertama: bagaimana banyak komputer tetap **benar dan hidup**.

Paruh kedua: bagaimana banyak prosesor bekerja **secepat mungkin**.

Keduanya bertemu di sistem nyata hari ini (*AI cluster*, platform data, layanan berskala jutaan pengguna) yang harus cepat **dan** tetap hidup ketika sebagian rusak.

Bekal ke depan

Fondasi ini sudah Anda pakai sepanjang semester di Data Engineering: *partition*, *shuffle*, *columnar*, *replication*, dan Kafka. Bawa kebiasaan terpenting dari mata kuliah ini: **ukur dulu, baru berpendapat**.

Selamat mengerjakan

Cepat itu bagus. Benar dan terukur itu wajib.