Teljesítmény-optimalizált Szoftverek - Performance-optimized computing

Kinek ajánljuk ezt tárgyat?

- Akiket inspirálnak a 'penge' -azaz minél gyorsabb, kisebb és hatékonyabb- szoftverek
- Akik ezért nyitottak az alapos tervezésre, gondos mérésekre, kitartó kísérletekre.
- Akik szeretnék megérteni és maximálisan kihasználni a modern processzorokat és hardvereket.
- ... akik többre tartják magukat, minthogy 'tucat-szoftverek tucat-programozói' legyenek.

A kurzus célja:

- A jövő szoftver-tervezőiként a hallgatók tudatosan vegyék figyelembe a programok teljesítményét és lábnyomát.
- Észleljék az egyes feladatok tervezése és megoldása során a performancia-kritikus részeket, problémákat és csapdákat.
- Reálisan tudják becsülni és megmérni az erőforrás igényeket.
- A teljesítményt leginkább meghatározó részeknél lehetőség szerint jobb karakterisztikájú - egyúttal sok esetben egyszerűbb - megoldásokat javasoljanak.
- Ismerjék a korszerű programozási technológiákat, így például a többszálú és elosztott szoftver-architekturákat, és fel tudják mérni, hogy ezeket mely esetekben érdemes alkalmazni.
- Hitelesen tudják összevetni a teljesítménybeli javulást az eredeti megoldással, illetve a készen elérhető, kereskedelmi vagy nyílt forrású megoldások teljesítményével.

Az oktatás menete:

A hallgatókkal az órák során elemzünk néhány tipikus feladatot és technológiát (pl. egyszerű string és mátrix feldolgozást, adattárolási opciókat, hálózati kommunikációs és üzenetkezelő/küldő rendszereket stb.), és megvizsgáljuk, hogy konkrét, specifikus feladatokra gondosan megírt saját kód mennyire tudja felvenni a versenyt a kész könyvtárakkal, keretrendszerekkel, szerver-szoftverekkel. Lesznek esetek, ahol a saját fejlesztés drámai javulást hoz, és lesznek olyanok is, ahol éppen a kész megoldás a gyorsabb, vagy a különbség nem elég szignifikáns a fejlesztési munka és egyéb szempontok (pl.: biztonság, rugalmasság, hordozhatóság) tükrében. Azt is vizsgáljuk majd, hogy egy saját fejlesztésű, vagy egy egyszerűbb, minimalista nyílt forrású komponens mennyire tudja redukálni egy-egy alkalmazás komplexitását.

A kurzus fő programozási nyelve a C/C++ -ez a bemutatott példák, kísérletek forrásnyelve-, de itt-ott még a generált gépi kódba is belenézünk. A hallgatók által választható feladatok egy része viszont más nyelveken (Java, C#, Python) is megoldható.

A tantárgy félévét két szakaszra osztjuk:

- Az első felében különböző elterjedt hardver és szoftver megoldások -pl. CPU cache, adattárolók, hálózatok, köztes szoftverek- általános bemutatásával fogunk foglalkozni, és ismertetünk bizonyos optimalizálási lehetőségeket, módszereket, és megvizsgáljuk ezek alkalmazatóságát, korlátait. A hallgatókkal közösen összegyűjtjük és megfogalmazzuk a szoftverek szabatos összevetésének elvi követelményeit, és az eredmények szabatos dokumentálásának módját.
- A kurzus negyedik hetében a hallgatók választanak maguknak egy-egy témát, elsősorban az általunk felkínáltak közül. A félév második felében az órák során konzultálunk a hallgatókkal a tervezett és folyamatban levő munkájukról.

A félév során két vendégelőadót is tervezünk meghívni, akik megosztják ipari tapasztalataikat, szempontjaikat, kb. 30-50 perc időtartamban.

Témakörök:

- A szoftver-optimalizálás szükségessége, és általános módszerei. Optimalizálási lehetőségek különféle nyelveken és környezetekben. Pozitív és negatív esettanulmányok.
- Programnyelvek (C/C++, Java, Python) teljesítményének összehasonlítása egyszerű, tipikus feladatok esetén. C fordító optimalizálási szintjei és opciói, ezek hatásának vizsgálata. Profile vézérelt optimalizálás. Java VM-ek különbségei, AOT/JIT
- Hardver architekturák lehetőségei, cache és lapozás-barát kódolás, ezek hatásának vizsgálata. Linux kernel és userspace programok teljesítményének különbsége, Linux Data Plane Development Kit.
- Fix és lebegőpontos aritmetikai, és matematikai műveletek implementációja, relatív sebessége.
- Hálózati kommunikáció fajtái, és teljesítménye különféle protokollok ill. kliens és szerver szoftverek esetén. Biztonságos (pl. TLS) kommunikáció hatása a teljesítményre, kulcs cserék és titkosító algoritmusok teljesítménye.
- Adatbázisok, üzenetkezelő és elosztott tároló rendszerek teljesítményének optimalizálása, gyakorlati limitek, skálázhatóság felső határai. Az adatbiztonság, redundancia, titkosítás és a teljesítmény összefüggése. Két- és többrétegű alkalmazások.
- Háttértárolás sebessége, korlátai, SSD és HD storage technológiák, cache és buffer stratégiák, ill. ezek hatása a programokra.

Számonkérés és értékelés:

A tárgy értékelése egyetlen, a hallgató által választott -de az oktatóval egyeztetett tartalmú- beadandó feladat alapján történik, melyet önállóan kell elkészíteni majd bemutatni. A hallgatói munka jellemzően tartalmazza egy adott informatikai feladathoz már létező piaci megoldás(ok) felkutatását, kipróbálását, valamint annak egy önálló, saját megoldását. Ezeken a kurzuson

megfogalmazott módszertan és szempontok szerint méréseket kell végezni, ezeket jegyzőkönyben összegezni. A saját megoldás elkészítése során törekedni kell a különböző aspektusok szerinti optimalizálásra, és ezeket részletesen be kell mutatni. A hallgatók a félév utolsó óráiban mutatják be az elvégzett munkáikat és azok tanulságait.

Irodalom:

· G. Hager, G. Wellein: Introduction to High Performance Computing for Scientists and Engineers, CRC Press, ISBN 978-1-4398-1192-4, 2011.

· R. Robey, Y. Zamora: Parallel and High Performance Computing, Manning, ISBN 9781617296468, 2021.

· G. Mashaki: The Art of High Performance Computing for Computational Science, Vol. 1: Techniques of Speedup and Parallelization for General Purposes, Springer, ISBN 978-9811361937, 2019

Performance-optimized computing

This course is recommended for:

- Those who are inspired by "nimble" - i.e. fast, small and efficient- software
- Those who are ready for detailed design, accurate measurements, unbiased experiments.
- Those who wish to understand and leverage the performance features of modern CPU-s and the surrounding hardware.
- …Those who wish to become more than just ordinary programmers of run-off-the-mill software.

Course objectives:

- Future software engineers shall deliberately consider the performance and footprint of the software they develop.
- It is important that they detect and identify the performance-critical sections of all software they design or develop.
- They should realistically estimate and accurately measure computer resource requirements.
- Thy should develop software with better characteristics -and often also with less complexity - for performance-critical sections.
- They shall be preraped to use advanced software architectures, e.g. multi-threaded or distributed solutions, and to measure their effectiveness.
- They shall execute unbiased comparisons to assess the improvement brought by their work, and if possible, also compare with solutions available from other sources e.g. commercial or open source software.

Course approach and prerequisites:

During this course we will analyze a few typical problems and techologies (e.g. string and matrix processing, options for storing data, network communications and messaging, etc.) and think about how some specific, custom developed software would compare to generic libraries, frameworks or server software which are readily available. We will see some cases where custom software will bring impressive improvement, while in other cases the differnce is not significant to justify the development work or the impacts on other characteristics (e.g. maintenance, security, portability, etc.). We will also see how some clever custom-developed code -or some minimalistic open-source library- can reduce the complexity and the footprint of the overall solution.

The main programming language used in this course -i.e. the language of most examples and experiments, is C/C++, but sometimes we will also look at the machine code generated. Many of the selectable individual work assignments can be done in other languages (e.g. Python, Java, C#) as well.

The semester will have 2 phases:

- The first phase will present some popular hardware and software technologies - e.g. CPU, cache, storage technologies, networks, middleware-, and generic optimization techniques, with their advantages, useability and also their limitations. Together with the students, we will formulate the best practices for credible and unbiased performance bechmarking, and how to document the results of such evaluations.
- Students will pick pick individual project work in the 4th week of the semester. From about the 7th week, we will do regular consulting with students on their work planned and in progress.

We will also invite two expert guests to present (in 30-50 minutes) their view and case studes of performance optimisation in the software industry.

Main topics discussed:

- Why software optimization makes a difference? Generic methods and optimization fetures in various languages. Positive and negative case studies.
- Comparing the performace of programming languages (C/C++, Java, Python) used for frequent and simple problems. C compiler optimiziation levels and options, and their impact. Profile-driven optimization. Java VM-s, and their characteristics. AOT/JIT
- Hardware architectures to improve performance. Cache- and paging-friendly programming, and their effect. Considerations of Linux kernel vs. userspace code. Linux Data Plane Development Kit.
- Relative speed of machine code instructions, e.g. fixed fixed and floating point operations.
- Types of network communication an their performance for various protocols, and client/server software. Impact on secure communications (pl. TLS), and encryption algorithms on performance.
- Performance optimization of databases, messaging middleware, an distributed storage. Practical limits and cielings for scalability. Interdependency of data protection, redundance and performance. 2- and N-layer applications.
- Speed and limits of permanent data storage. SSD and HD , caching, buffering and their influence on software characteristics.