Literaturanalyse

- Ohne Vorhersagemodell (nur Messungen) oder mit Vorhersagemodell (welches trainiert werden muss)
- Mit Modell:
  - Würde den Umfang der Arbeit sprengen
  - Lieber auf die Validierung eines existeierenden Modells mit neueren Methoden konzentrieren
- Architektur: Ein Ansatz kann entweder auf eine Spezial- oder eine Allzweckarchitektur abzielen
- Granularität: Die Ansätze unterscheiden sich in der Granularität, mit der sie messen können
- Annahmen: Die Annahmen, die Ansätze über das gemessene Programm oder seine Umgebung machen
- Prof. Siegmund hat noch keine auswertbaren Daten für seine Messungen veröffentlicht.

Erkenntnisse

- Es gibt kein fertiges Standard-Framework, welches man einfach verwenden kann.
- Es wird viel mit wenig Strom verbrauchenden Prozessoren gearbeitet (ARM, Mikrocontroller-Architekturen, x86 mit geringem Energieverbrauch)
- Messungen mit Serverprozessoren wie der Intel Xeon Reihe kommen fast gar nicht vor.
- Ich würde mich auf x86-Laptop/Desktop/Server-Prozessoren beschränken, da ich dort die meisten Erfahrungen habe
- Direktes Messen vom Stromverbrauch von Methoden ist nur auf primitiven Mikrocontrollern möglich
- Es besteht die Möglichkeit den Stromverbrauch des ganzen Computers, der einzelnen Komponenten zu bestimmen oder nur den der CPU. Man misst entsprechend direkt von der Hardware, nutzt den Stromverbrauch der Batterie oder nutzt ein Modell zur Schätzung
- Der modernste Weg ist die Energieverbrauchsmetriken der CPU zu verwenden über RAPL (Running Average Power Limit). Das war bis vor kurzem nur bei Intel Prozessoren möglich, ist aber auch in modernen AMD Prozessoeren enthalten.
  - Blendet andere Komponenten aus, aber solange die Applikation nicht grafisch (Display und GPU) ist, ist es trotzdem repräsentativ
  - Speichernutzung kann anhand des Energieverbrauchs des DRAM-Speichercontrollers ermittelt werden
  - Messungen werden rund aller 10-100ms aktualisiert

  - Die Energieangabe gilt für den GESAMTEN Prozessor, andere Programme bzw. Grundlast müssen berücksichtigt werden!
- Energiemodelle für Java-Bytecode existieren
  - wurden aber nie mit Hardware validiert
  - Methodengranularität scheint möglich zu sein
  - Für Mobile Geräte
  - oft nicht fortgeführt und kein Quellcode vorhanden
- Energiemodelle für C
  - basieren darauf eine Methode oft hintereinander auszuführen, sodass eine Energiemessung über Zeit erfolgen kann
  - gibt auch Ansätze über den AST
  - Es werden auch unter AOP einzelne Sortieralgorithmen verglichen
- Alle Ansätze mit beigelegtem Quellcode leiden unter der Tatsache, dass Programme nach Veröffentlichung des Papers nicht mehr weiterentwickelt werden. Ältere Benchmarks sind häufig selbst gebaut und können so nicht reproduziert werden. Deshalb möchte ich möglichst viel Standard- bzw. FOSS-Software verwenden.
- Angaben verbrauchter Joules kann man nicht übernehmen, da der Energieverbrauch eines Prozessors vom Modell zu Modell unterschiedlich ist. Validierung muss so über das Verhältnis des Verbrauchs verschiedener Instruktionen zueinander erfolgen oder mit einer anderen Mehtode wie das Messen des Batteriestandes
- Alle bisherigen Ansätze über RAPL messen den Energieverbrauch der gesamten CPU zum Zeitpunkt x, Interferenzen von anderen laufenden Programmen kann nicht ausgeschlossen werden. Scaphandre ist ein auf RAPL aufbauendes Framework welches den Energieverbrauch der Prozesse trennen kann, d.h. man kann einen einzelnen Prozess betrachten. Scaphandre ist neu auf GitHub und hat noch keine Publikationen, ich würde meine Arbeit auch gerne nutzen darauf aufmerksam zu machen.

Zur Verfügung stehende Hardware

- Linux-Laptop mit Intel-Prozessor
- Windows-Desktop mit AMD-Prozessor
- Linux-Server mit Intel-Prozessor

Messsoftware

- Intel RAPL - Running Average Power Limit
  - https://01.org/blogs/2014/running-average-power-limit-%E2%80%93-rapl
  - Bibliothek um eine Prozessor-API auszulesen
  - Stromverbrauch eines Prozessors wird in einem Zeitraum in Jouls gemessen
- Scaphandre

  - https://github.com/hubblo-org/scaphandre
  - in Rust geschrieben
  - Stromverbrauch eines isolierten Prozesses mit RAPL messbar
  - sollte auch auf AMD und zukünftig ARM funktionieren
  - Schnittstellen sind noch sehr primitiv bzw. nicht vorhanden
  - JSON output, ist aber grausam aufgebaut
  - Exporter für Monitoring-Software wie Prometheus vorhanden
  - Ist ein Open-Source Projekt, welches auch relativ beliebt ist
- VTune Profiler
  - https://software.intel.com/content/www/us/en/develop/tools/oneapi/components/vtune-profiler.html
  - Profiler für C/C++/Fortran und Java von Intel
  - sehr umfangreich, hilft beim Finden von Hotspots, Bottlenecks etc.
  - Ein Deep Dive hier wäre aüßerst interessant, aber auch nur das Verwenden eines Programms
  - funktioniert **nur** mit Intel-Prozessoren, AMD wird nicht unterstützt
  - sollte auf einem seperaten Rechner, als dem der gemessen wird, ausgeführt werden.
- AMD uProf
  - https://developer.amd.com/amd-uprof/
  - AMD Äquivalent zu VTune, sehr primitiv
  - würde ich nicht empfehlen
- VisualVM
  - Profiler für Java, zeigt kumulierte verbrauchte CPU-Zeit pro Funktion an
  - Gut um einen Überblick zu bekommen und mit Kieker genauer nachzuhacken
  - Open Source Alternative zum JProfiler
- Kieker
  - https://kieker-monitoring.net/
  - Für Feinmessungen, gut einstellbar, Integration mit dem Dashboard bereits vorhanden
  - Bereits Erfahrung vorhanden durch Kieker-Integration ins jQAssistant
- https://gitlab.com/MarcoCouto/c-lem https://gitlab.com/MarcoCouto/serapis
  - Um Energiekosten von C-Instruktionen zu ermitteln
  - Wurde nach Veröffentlichung des dazugehörigen Papers nicht fortgeführt
  - Keine Dokumentation, von daher schwierig durchzuführen
  - Funktioniert nicht auf meiner Hardware, so wie es aussieht wurde der Code auf den PC des ursprünglichen Entwicklers ausgelegt

  - War das vielversprechensde von allem, man kann ggf. das Prinzip neu implementieren

- powerapi
  - https://github.com/rouvoy/powerapi
  - Energieverbrauch auf Prozess-Level
  - kein Update seit 5 Jahren
  - stattdessen Powertop verwenden
- Powertop
  - https://01.org/powertop/
  - Nutzt Energieverbrauch der Batterie
  - funktioniert, trennt Prozesse voneinander
  - Perfekt geeignet zur Validierung
- Linux Bordmittel
  - power_now
  - powercap
- Valgrind
  - Binary (C/C++/Fortran) Profiler
  - Methodengranular
  - Real und CPU-Zeit
  - Eigene Visualisierung
  - Höherer Overhead -> Längere Laufzeit des Programms
- gperftools
- Prime95
  - Zum Auslasten aller CPU-Kerne, damit man den maximalen Stromverbrauch des Prozessors ermitteln kann
  - Funktioniert derzeit nur bedingt gut, da nicht alle Teile des Prozessors ausgelastet werden
- Die Evaluierung der gezeigten Ansätze mache ich über das nächste Wochenende.

Programmiersprachen

- Wenn Genauigkeit auf Methodenlevel erziehlt werden soll, muss zwangsläufig eine Eingrenzung auf eine Programmiersprachenfamilie erfolgen
- Ich habe aus meiner Sicht genug Hintergrundwissen in Java und C um dies zu bewerkstelligen, bei anderen Programmiersprachen fehlt mir zu sehr der Kontext
- Java
  - Garbage Collection verursacht Hintergrundrauschen

  - Threading-Modell verursacht unter Linux ggf. Schwierigkeiten bei der Messung
- C/C++/Fortran
  - Erzeugt plattform-spezifische Binaries
  - Kein dynamischer Overhead durch Garbage Collection
  - Genaue Angabe von Optimierungsoptinen nötig um ggf. abwichende Ergebnisse zu verwenden

Fallstricke

- Hyperthreading
- Frequenzanpassung der CPU
- x86, die Vielzahl von zusätzlichen Befehlssatzerweiterungen und das Verhalten der Caches untereinander sind so komplex, dass man darüber eine eigene Arbeit schreiben könnte. Insbesondere kann man keine standfesten theoretischen Aussagen treffen (da Closed Source).
- Sind Größen korreliert?
  - e.g. Laufzeit/CPU-Zeit und Energieverbrauch
- Sehr kurze Methoden kann man schlecht berücksichtigen, wenn diese kürzer als ein Testintervall ist.
- Tendenziell sind einige wenige Funktionen für den Großteil der verbrauchten CPU-Zeit verantwortlich.
- Die Rechnungs- und Speicherkomplexität von Funktionen spielt beim Energieverbrauch auch eine Rolle. Man kann dieses Problem umgehen, wenn weit verbreitete Benchmarks verwendet werden, dadurch wird halt immer der selbe Code ausgeführt.

Derzeitiger Plan

C-basiert

- Benchmarks aus `Statically Analyzing the Energy Efficiency of Software Product Lines` validieren mit `Scaphandre`
  - Nutzt verschiedene C-SPLs und Community Benchmarks
  - Rechnet anhand eines Modells und RAPL die Energiekosten aus, diese werden mit `Scaphandre` validiert
- Energiekosten werden nochmal zusätzlich anhand einer Messung des Batteriestandes meines Laptops validiert
- C-Funktion und Instruktionskosten einer SPL durch C-LEM messen und mit Metapher visualisieren
- getaviz hat keine C-Unterstützung
- jqassistant hat nur in alten Versionen C-Unterstützungen
- https://gitlab.com/MarcoCouto/c-lem

  - Am vielversprechendsten auf den ersten Blick
  - funktioniert nicht mit meinem Prozessortyp
  - müsste angepasst werden
  - Anpassung schon “toter” Software ist aus meiner Sicht wenig zielführend

- C-basierter Ansatz muss so anders angegangen werden
- Der einfachste und primitivste Ansatz wäre typische C-Routinen wie Sortieralgorithmen etc. zu vergleichen indem man diese lange laufen lässt und den Energieverbrauch, Laufzeit und verbrauchte CPU-Zeit derweil misst. Der Visualisierungsteil gestaltet sich dabei noch schwierig.
- VTune und Valgrind haben hier auch Mittel zum Messen und Visualisieren von Performance, aber die Lösung für Energieverbrauch gibt es halt nicht.

Java basiert

- Eine der Applikationen die Prof. Siegmund in seinen Papern häufig verwendet nehmen.
- Mit VisualVM profilen um sich einen Überblick zu verschaffen
- Kieker misst standardmäßig vergangene Realzeit, nicht die vergangene CPU-Zeit
  - Muss ggf. angepasst werden
- Verbraucht Energie derweil mit Scaphandre messen
- Wenn Realzeit bzw. CPU-Zeit mit Energieverbrauch korrelliert sind kann man es darüber versuchen

Visualisierung

- Da wir hier nur auf einer Ebene messen, wird die hierarchische Visualisierung schwierig.

Wichtigste Literatur

- Methodological Guidelines for Measuring Energy Consumption of Software Applications
  - Grundlagen der Methoden zum Messen
  - Werde ich zur Einordnung der verwendeten Methoden nehmen
- Survey of Approaches for Assessing Software Energy Consumption
  - Existierende praktische Methoden
- Characterizing the energy consumption of data transfers and arithmetic operations on x86-64 processors
  - Erklärt wie Störfaktoren (dynamische Frequenz etc.) deaktiviert werden können
- How to measure energy-efficiency of software: Metrics and measurement results
  - Ansatz zum Messen in Joule

  - Kombiniert Black- und WHite-Box-Modelle
- Automating Energy Optimization with Features
  - Frühes Paper vom Prof. Siegmund zum Thema
  - Etabliert grundlegende Zusammenhänge
  - Feature-Oriented Programming
- Statically Analyzing the Energy Efficiency of Software Product Lines
  - C basierend
  - Nutzt abgeleiteten AST (C-Lem, Serapis) und RAPL um Energieverbrauch von Komponenten (Statements, Methoden, ganzes Programm)
  - https://gitlab.com/MarcoCouto/c-lem
  - https://gitlab.com/MarcoCouto/serapis
- Estimating Mobile Application Energy Consumption using Program Analysis
  - eLens Framework, keine Weiterentwicklung, kein Quellcode auffindbar
  - Für Smartphones
  - Nutzt software environment energy profile (SEEP) vom Hersteller
- Component-Level Energy Consumption Estimation for Distributed Java-Based Software Systems
  - often cited, uses Java
  - Uses Computational, Communication, Infrastructure Energy Consumption for distributed systems
  - Energy consumption estimation framework including static and runtime estimation
  - Evaluation over experiment (achives method level granularity)
  - only works with Interpreter-JVMs, not with modern JIT-JVMs
  - states that execution time and energy consumption are not correlated
- CPU Energy Meter: A Tool for Energy-Aware Algorithms Engineering
  - Misst Energieverbrauch via RAPL
  - Vergleicht Sortieralgorithmen
  - https://github.com/sosy-lab/cpu-energy-meter
- GreenOracle: Estimating Software Energy Consumption with Energy Measurement Corpora
  - Für Smartphones
  - Basiert auf CPU-Auslastung und Anzahl von System-Calls
- A Model-based Framework for the Analysis of Software Energy Consumption
  - Erweitert existierende Hidden-Markow-Modelle um Energiekosten
  - nutzt jRAPL + LTS-Analyse Tool LoTuS (nicht mehr verfügbar)
  - Graphenbasiert
  - Sagt aus, dass Java viel Execution overhead (JVM) hat, und dadurch die Energiemessungen nicht wirklich korrekt zugeschrieben werden können
  - Lässt Grundlast außen vor
  - Long und Double verbrauchen mehr Energie
  - kein angehängter Code