Patterns in Machine Learning

Application of Pattern Based Parallelization to Industrial ML Use Cases

Holger Schöner, Michael Roßbory

+43 7236 3343 816
holger.schoener@scch.at
www.scch.at
Machine Learning
Use Cases
ML Use Cases

Algorithms

Processes

Data

- Intelligence flows back into machines
- Extraction and storage of proprietary machine data stream
- Data sharing with the right people and machines
- Machine-based algorithms and data analysis
- Secure, cloud-based network
- Instrumented industrial machine
- Industrial data systems

graphlab.org
ML Use Cases

Parallelization of Machine Learning Use Cases

- **High parallelization potential**
  Independent models/architectures/algorithms in parallel
  Linear algebra, graph structures, ...

- **Rising amounts of data**
  Need for more computational power
  Need for more memory
ML Use Cases

- Parallelization of Machine Learning Use Cases
  - Target architectures
    Used in different software systems at multiple customer sites
    → flexibility wrt. target architecture, power of system
  - Code
    Legacy Code, or sequential then parallel implementation
    To ease check for correctness, scaling and debugging
ML Use Cases – Cutting Stock Problem
ML Use Cases – Cutting Stock Problem

General Problem: One dimensional cutting stock problem

- Discrete linear optimization
- Optimize Resources, complying with various constraints
- Computationally very intensive

wikipedia.org/wiki/Cutting_stock_problem
ML Use Cases – Cutting Stock Problem

**Specific Use Case**

- Equivalent to make-to-order reel and sheet cutting in paper mill
- Set of sheets with different dimensions
- Minimize tool changes, minimize waste
- Comply with available reels and tools

wikipedia.org/wiki/Cutting_stock_problem
Cutting Stock Problem – Sequential

- Initially calculate one admissible solution
- Fill pool with multiple copies of that
- Cycle: pick one solution and feed into optimization chain, then evaluate collector function
- Optimization chain: Several steps of interleaved optimizations and constraint enforcements
Cutting Stock Problem – Sequential

- Collector: evaluate solution and either
  - Put (some copies of) new solution back into pool, or
  - Dismiss solution (if not improved)

- Termination
  - Maximum number of cycles
  - Objective function below target threshold
  - No improvement after certain number of cycles
Cutting Stock Problem – Parallel Port (Pipeline)

- Use ParaPhrase/Fastflow pipeline pattern (skeleton version)
- First pipeline stage (emitter): pick some solution from pool
- Further stages of pipeline correspond to optimization steps
- Last pipeline stage (collector): same as sequential collector
- Number of pipeline stages (worker threads) equal to number of optimization steps + 2
Cutting Stock Problem – Parallel Port (Farm/Pipeline)

- Use ParaPhrase/Fastflow farm and nested pipeline patterns
- Each farm worker is a complete optimization pipeline
  - As in Parallel Port (Pipeline), without collector stage
- Collector is shared by all farm workers, could be bottleneck
- Number of cores should be multiple of pipeline stages + collector
Cutting Stock Problem – Parallel Port (Farm/Pipeline/Feedback)

- Use ParaPhrase/Fastflow farm and pipeline patterns, with feedback
- Add objective function evaluation stage to pipelines
- Collector only has to compare objective function values
- Feedback mechanism replaces intermediate solution pool
- Optional load balancer
ML Use Cases – Waste Water Processing
Dependency Analysis

- Identify influences on carbon contamination of processed water

- Methods
  - Graphical Lasso
  - Granger Causality

- Identify graph of dependencies and causal relations between input, control, sensor and output variables

statweb.stanford.edu/~tibs/ftp/glasso-bio.pdf
www.stanford.edu/~hastie/Papers/glassoinsights.pdf
### ML Use Cases – Waste Water Processing

- **Graphical Lasso**
  Find structure by L1-regularized (sparse) covariance estimation

- **Granger Causality**
  Does X contain useful information for predicting Y?

\[ y(t) = \sum a_k y(t - k) + \sum_{i,k} b_{i,k} x_i(t - k) \]

Implemented by Graphical Lasso on time-lagged features → Drastically increased matrix size
Waste Water Processing – Parallel Port (Farm)

- Use ParaPhrase/Fastflow farm pattern (accelerator version)
- Graphical Lasso:
  5 nested loops, 3 parallelizable, results for topmost parallelization
- Split Covariance Matrix into connected components, work on these independently
Waste Water Processing – Parallel Port

- **Issues**
  - Best results if size of largest components is roughly equal (and there are enough of these)
  - Split into components depends on regularization:
    - Low $\lambda_1$ – only one component, potentially intractable
    - High $\lambda_1$ – only small components, computationally simple

- **Potential Approach**
  - Parallelization (alternatives) of further loop levels
  - Data/regularization dependent switch between variants
  - Parallelization of innermost loop limited due to weak dependencies between features/iterations
  - Initial results indicate speedups of 2-4 realizable
Experiences
Porting Legacy Code

- Refactoring for Hygiene
  - Determine race conditions and shared resources
  - Implement proper Copy/Assignment operators

- Comparison with Sequential Version
  - Check for correctness
  - Check for scaling

- Optimization and Parallelization
  - With our legacy code, coding roles are/were still distributed

- Porting Approach
  - Patterns / Parallel structure straightforward, for our use cases
  - Really good performance needs lots of evaluation runs and comparison of variants
  - Time consuming (compared to OpenMP) without refactoring tool, relatively close with refactoring tool, ParFor
Parallelization Decisions

- **Skeleton or Offloading Approach**
  - Skeleton approach: clearer structure
  - Offloading: (sometimes?) less code changes
  - Variants with / without collector

- **Default / Explicit / No Pinning**

- **Memory locality**
  - Copy local elements of workers from shared data
  - Or work in shared memory, avoiding copy costs?
  - Not straightforward; depends on memory access patterns and pinning strategy

- **Generally Good Performance**
  - As opposed to detailed architecture tuning, for few code versions
  - Needs lots of benchmark runs, access to variety of machines
  - Currently, code variants not completely avoidable
Comparison of Approaches
Patterns vs. OpenMP

- **Coding / Porting Time**
  - OpenMP **very simple and fast** (for simple parallel approaches)
  - ParaPhrase without refactoring **quite verbose**
  - ParaPhrase with refactoring and/or ParFor **close to OpenMP**
  - Both can require quite some **time for tuning and code cleanup**

- **Flexibility**
  - OpenMP **needs compiler support**
  - ParaPhrase only **needs C++ compiler, threading library**
  - OpenMP for **multicore machines** *(heterogeneous possible now)*
  - ParaPhrase for **multicore, heterogeneous machines** *(distributed possible)*

- **Maintenance, Documentation, ...**
Exercises
Cutting Stock Problem

■ **Framework**
  - Generic framework for different kinds of optimization problems
  - Provided objective function and constraints correspond to cutting stock problem
    - Set `folder` variable in `SheetOptimization.cpp` to `../tf_pc/`

■ **Build**
  - In folder
    - `SCCH/DiscreteOptimization/OptimizationUC_/src_opt_fw/build`
  - `cmake`
  - `make`

■ **Run**
  - From `src_opt_fw` folder, call `build/bin/SeriesOptimization`
  - **Cmd option** `-obj`: number of initial objects in solution pool
  - **Cmd option** `-len`: number of optimization steps in pipeline
  - **Cmd option** `-t`: Duration of interleaved optimization steps [ms]
## Cutting Stock Problem

### Parallelization
- Make pipeline out of steps in `StepByStep::run()` in file `src_opt_fw/src/OptiFramework/StepByStep.h`
- Use structure already provided in `SCCH/DiscreteOptimization/OptimizationUC_FF_template`

### Project Structure
- `Src/SheetOptimization.cpp` – main program
- `Src/core` – unimportant details, heuristic optimization functions
- `Src/OptiFramework/StepByStep.h` – OptimizationStepChain (iterations), parallelized also pipeline worker (collector) `svc()`
- `Src/OptiFramework/WorkpieceProcessor` – new pipeline worker, necessary for `svc()` function
- `Src/OptiFramework/WorkpieceEmitter` – new emitter for pipeline, necessary for `svc()` function
Graphical Lasso

- **Parallelization**
  - Two executables, `gelnet_seq_summer-school` and `gelnet_ff_summer-school` (extended version)
  - Some structure is already provided in `gelnet_cov_ff.cpp`, where `t_gelnet_ff` should become the parallelized version of `t_gelnet` (in `gelnet_cov_seq.cpp`)
  - The loop over connected components in `t_gelnet()` should be parallelized using the farm pattern (optionally comparison with ParFor?)

- **Build**
  - `make`

- **Run**
  - Look at `sim.sh`, calling executables with standard data/options
  - First cmd line there is quickest (to check for correctness), later ones take (much) longer (to provide work for multiple workers)

- **Plotting**
  - Call `python eval_plots.py` after copy&pasting relevant executable output
Graphical Lasso

Project Structure

- `gelnet_ff_summer-school.cpp` – main program, no changes necessary
- `gelnet_cov_ff.cpp` – only file to modify, contains `t_gelnet_ff()` as parallelized `t_gelnet()` from `gelnet_cov_seq.cpp`, also contains some prepared data structures and functions for parallelization
- `gelnet_cov_seq.cpp` – sequential version of code, use to copy&paste code blocks for the parallel version (only `t_gelnet()` is relevant)
Thank you

www.scch.at
www.paraphrase-ict.eu